CREACIÓN DE UN ENTORNO PARA HACKATONES DE CIBERSEGURIDAD

CREATION OF A FRAMEWORK FOR CYBERSECURITY HACKATÓNS

TRABAJO FIN DE GRADO
CURSO 2024-2025

AUTORES Y NOTAS
MANUEL LOURO MENESES 9,6
FEDERICO GUILLERMO LÓPEZ MERINO 8,4
PABLO SEVILLANO RUIZ 6,6

DIRECTOR
MARCOS SANCHEZ-ELEZ MARTIN

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

CREACIÓN DE UN ENTORNO PARA HACKATONES DE CIBERSEGURIDAD

CREATION OF A FRAMEWORK FOR CYBERSECURITY HACKATÓNS

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTOR

MANUEL LOURO MENESES

FEDERICO GUILLERMO LÓPEZ MERINO

PABLO SEVILLANO RUIZ

DIRECTOR

MARCOS SANCHEZ-ELEZ MARTIN

CONVOCATORIA: JUNIO 2025

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

7 DE MAYO DE 2025

DEDICATORIA

A todos nuestros compañeros, amigos y familia que que nos han acompañado durante de este camino.

AGRADECIMIENTOS

A Marcos por habernos descubierto e iniciado en el gran mundo que es la Ciberseguridad. Al decano Luis Hernández Yánez por habernos ayudado a que este grupo de 3 haya sido posible.

RESUMEN

Creación de un Entorno para Hackatones de Ciberseguridad

Este Trabajo de Fin de Grado presenta el diseño e implementación de un entorno controlado y seguro para la celebración de hackatones de ciberseguridad en laboratorios docentes. Estos eventos, al implicar la instalación de múltiples herramientas especializadas y la participación simultánea de numerosos usuarios, pueden exponer la infraestructura a diversas vulnerabilidades. Por ello, se plantea un sistema que permita a los organizadores supervisar de forma eficaz el uso de los equipos y garantizar un entorno fiable y auditable.

El entorno desarrollado incluye un conjunto de herramientas destinadas a monitorizar y auditar en tiempo real, con una interfaz clara e intuitiva para facilitar su uso por parte de los docentes. Estas herramientas permiten detectar comportamientos sospechosos sin comprometer el rendimiento de los sistemas, contribuyendo así a preservar la integridad de la red durante el evento. Además, el proyecto fomenta el aprendizaje práctico en ciberseguridad dentro de un marco controlado y educativo, alineado con los principios del ethical hacking.

Palabras clave

Monitorización, Hackatón, Auditoría, Ciberseguridad, Automatización, Vulnerabilidad, Ethical Hacking, Educación.

ABSTRACT

Creation of a Framework for Cybersecurity Hackatóns

This Final Degree Project presents the design and implementation of a controlled and secure environment for the celebration of cybersecurity hackathons in educational laboratories. These events, involving the installation of multiple specialized tools and the simultaneous participation of numerous users, can expose the infrastructure to various vulnerabilities. Therefore, a system is proposed that allows organizers to effectively monitor the use of equipment and ensure a reliable and auditable environment.

The developed environment includes a set of tools aimed at real-time monitoring and auditing, with a clear and intuitive interface to facilitate its use by educators. These tools allow the detection of suspicious behavior without compromising system performance, thus helping to preserve network integrity during the event. In addition, the project promotes practical learning in cybersecurity within a controlled and educational framework, aligned with the principles of ethical hacking.

Keywords

Monitoring, Hackathon, Auditing, Cybersecurity, Automation, Vulnerability, Ethical Hacking, Education.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

Los laboratorios de la Facultad de Informática albergan una gran cantidad de aplicaciones y son utilizados por numerosos estudiantes, lo que puede generar vulnerabilidades de seguridad. Así, resulta fundamental que los estudiantes adquieran conocimientos prácticos sobre la protección de sistemas y detección de amenazas.

Para abordar estos desafíos, los entornos controlados como los hackatones de ciberseguridad se plantean como una oportunidad para experimentar y aprender, a la par que una buena estrategia para que los estudiantes puedan poner en práctica sus conocimientos en un entorno real y seguro.

Es evidente que, para garantizar correcta utilización de los laboratorios durante estos eventos es necesario utilizar herramientas que permitan la supervisión y auditoría de los equipos en tiempo real. Por consiguiente, el presente Trabajo de Fin de Grado (en adelante, TFG) surge con la motivación de dotar a los equipos de la facultad de un entorno preparado para realizar un hackatón orientado a la ciberseguridad, que permita a los participantes enfrentarse a este tipo de escenarios sin comprometer la integridad de los equipos ni la red de la facultad.

1.2 Objetivos

El objeto del presente Trabajo consiste en desarrollar una solución basada en el sistema operativo usado por los ordenadores de los laboratorios, incluyendo herramientas propias para auditar el sistema y monitorizar accesos sensibles en tiempo real. Se pretende así, fomentar el aprendizaje activo y demostrar la importancia de la preparación, detección y respuesta ante incidentes de seguridad que pudieran surgir en el entorno académico.

Concretamente, con todo lo anterior, se pretende:

En primer lugar, desarrollar una herramienta de auditoría capaz de analizar el estado del sistema antes y después de la realización del hackatón, pudiendo identificar las posibles alteraciones y/o comportamientos sospechosos. Asimismo, se pretende

también poder realizar una comparación entre el estado inicial y final del sistema mediante informes generados por la herramienta recién mencionada.

En segundo lugar, se pretende crear una herramienta de monitorización en tiempo real que pueda registrar, y alertar, sobre accesos a archivos sensibles definidos por el organizador del evento.

En tercer lugar, implementar una herramienta que defienda al sistema ante accesos no deseados por parte del participante, bloqueando su acceso y alertando al responsable.

En cuarto lugar, deberá configurarse y adaptarse el sistema operativo de los ordenadores de la facultad para conseguir el funcionamiento de un entorno seguro y controlado durante la celebración del evento.

En quinto lugar, se pretende también diseñar una actividad formativa introductoria dirigida a los participantes. Esta formación incluiría nociones básicas de ciberseguridad y la ejecución de un ataque controlado al bit SUID.

Por último, se quiere garantizar la persistencia y funcionamiento continuo de la herramienta de monitorización como un servicio del sistema.

1.3 Plan de trabajo

El desarrollo de nuestro proyecto ha seguido una metodología flexible y adaptativa, basada en la colaboración continua y la asignación de roles dentro del equipo. A lo largo del proceso, hemos implementado prácticas que se asemejan a metodologías ágiles, especialmente a un enfoque como Scrum [1], aunque sin una estructura formal rígida.

Desde el inicio del proyecto, nuestro equipo estaba compuesto por tres integrantes con diferentes responsabilidades: Uno ha sido responsable del desarrollo y programación de las herramientas. Otro estaba encargado de la documentación, recopilación de información y la redacción de la memoria. Finalmente, el tercero, se ha dedicado a la gestión del repositorio de GitHub, documentación técnica de las herramienta y desarrollo de una herramienta.

Si bien al comienzo del TFG la organización no estaba claramente definida, progresivamente establecimos una distribución de tareas más eficiente, permitiendo que cada miembro del equipo se especializara en un área específica.
La comunicación ha sido clave en el desarrollo del proyecto. Para coordinar nuestro trabajo, hemos utilizado un grupo de WhatsApp, donde se han discutido los avances y dudas diarias; y también hemos hecho reuniones semanales en nuestro servidor de Discord, donde hemos revisado el progreso y hemos establecido nuevas tareas.
Estas reuniones han funcionado como sprints semanales, en las que analizábamos el estado del proyecto y fijábamos los siguientes objetivos a completar. De esta manera, sin seguir un marco formal, hemos implementado una dinámica de trabajo iterativa y adaptable.
Para la gestión y desarrollo del proyecto, hemos estado empleado varias herramientas que han facilitado el progreso conjunto colaborando de manera telemática y síncrona.
Respecto a la *documentación*, hemos empleado Google Drive a modo de almacenamiento y compartición de documentos y recursos que hemos empleado para la redacción de la memoria, que ha sido posible gracias a Microsoft Word, en la que nos hemos ayudado de la funcionalidad de compartir el documento.
Respecto al código de las herramientas, hemos empleado el Visual Studio Code para desarrollar el código y de cara a su publicación en nuestro GitHub, donde hemos almacenado nuestro progreso por versiones y las herramientas en sí, hemos usado GitHub Desktop. Esta aplicación nos ha permitido ir publicando directamente el código sin hacer uso de comandos y de una manera muy sencilla. Finalmente, hemos utilizado VirtualBox para trabajar con el sistema Ubuntu indicado y para poder implementar el entorno que se describe en esta memoria. Esta aplicación nos ha facilitado, además de la ejecución de un sistema operativo sin necesidad de instalarlo en nuestros portátiles, la exportación de este entorno para poderlo emplear en futuros hackatones.

Para lograr una correcta organización y una distribución eficiente del plan de trabajo, el presente proyecto se ha estructurado en las diferentes fases que a continuación se detallan:

1. **Investigación preliminar**

   Comenzamos con una etapa de documentación e investigación para comprender el contexto de los hackatones de ciberseguridad, el concepto de las herramientas enfocadas a la ciberseguridad, entender cómo funciona el sistema operativo que hemos empleado y el lenguaje de programación que hemos usado. Esta fase permitió definir con claridad los objetivos técnicos y funcionales del proyecto.

2. **Diseño y desarrollo de herramientas**

   Partiendo de los requisitos establecidos en el punto siguiente, se diseñaron y programaron tres soluciones clave:

   - **Herramienta de auditoría del sistema**, para recoger y analizar configuraciones y acontecimientos críticos.
   - **Herramienta de monitorización**, que supervisa en tiempo real el estado de los servicios y recursos.
   - **Herramienta de defensa**, capaz de aplicar contramedidas automáticas ante detección de actividades anómalas.

3. **Pruebas y validación del entorno.**

   Se llevaron a cabo pruebas en entornos controlados para comprobar el correcto funcionamiento de cada herramienta. Luego se instalaron estas herramientas en un mismo entorno creando así la base para celebrar hackathones de ciberseguridad.

4. **Diseño y ejecución del evento**

   Partiendo del entorno base, se creó un escenario de prueba que servía como primera propuesta de hackatón, que permitía su despliegue en otros equipos. A continuación, se ejecutó un piloto en el equipo de un compañero. Las herramientas de auditoría se iniciaron al comienzo y al cierre del evento, mientras que las soluciones de monitorización y defensa funcionaron en tiempo real para validar su correcto comportamiento.

5. **Documentación final: Memoria**

   Para elaborar esta memoria se fueron recopilando todos los documentos generados durante la investigación y se describió detalladamente el proceso de desarrollo, así como el funcionamiento de cada herramienta. A continuación, se

documentaron los pasos seguidos para crear el entorno y llevar a cabo la prueba del hackatón. Finalmente, se presentaron los resultados obtenidos y las conclusiones derivadas del proyecto.

Para representar estas fases y el tiempo invertido, de una manera más visual, hemos incluido un diagrama de Gantt (Figura 1: Diagrama de Gantt) en el cual se muestra la planificación temporal de cada una de las tareas a lo largo del proyecto.

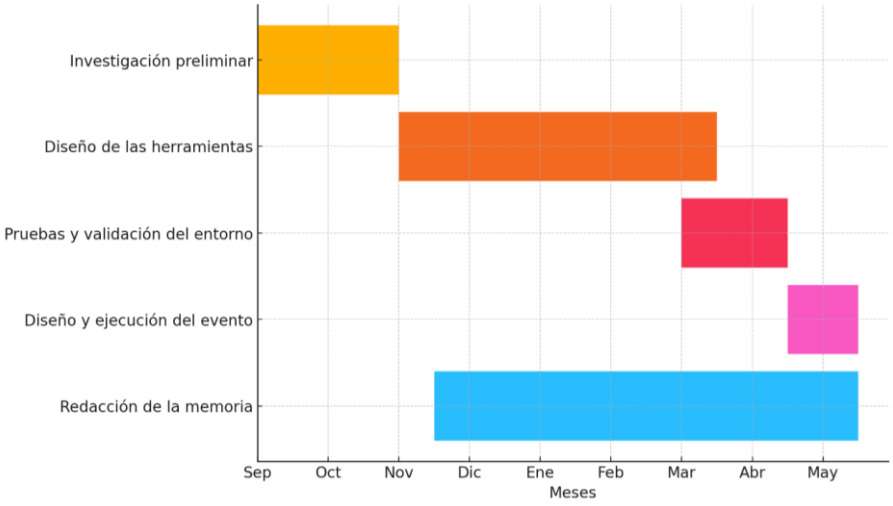

Figura 1: Diagrama de Gantt

Además de este diagrama, hemos querido hacer hincapié en mostrar cómo ha sido el avance del desarrollo, insistiendo más en cómo ha sido el progreso del diseño de las herramientas. Para ello, hemos optado por incluir a continuación un segundo diagrama (Figura 2: Diagrama de Gantt – Desarrollo herramientas) para hacer más visual este desarrollo.

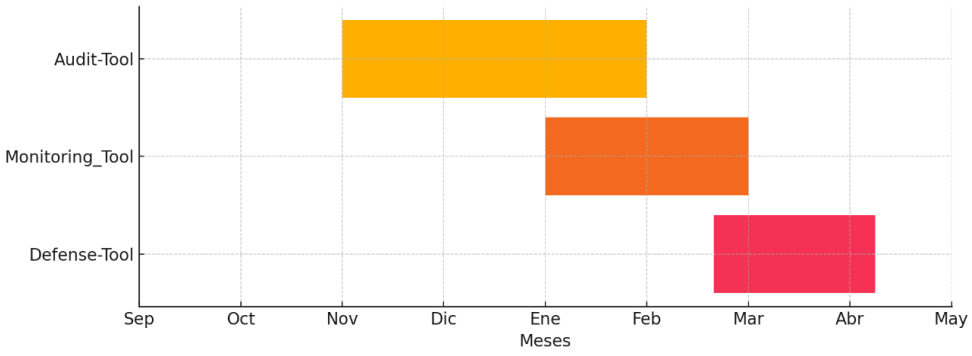

Figura 2: Diagrama de Gantt - Desarrollo herramientas

Como se puede apreciar, el desarrollo se fue solapando uno con el otro ya que temíamos no finalizar a tiempo todas las herramientas propuestas debido a las complicaciones que fueron surgiendo.

1.4 Investigación de requisitos

Cuando se trata del desarrollo de un proyecto técnico, la definición de requisitos es un paso fundamental para garantizar que el resultado final cumpla con su propósito de manera eficiente. Aquí se recogen los requisitos funcionales y no funcionales que planteamos, para llevar a cabo nuestro entorno, diseñado para funcionar en un sistema Linux con el propósito de celebrar hackatones.

La finalidad de esta investigación es establecer las características esenciales que debe cumplir el entorno que queremos implementar, asegurando que proporcione monitoreo continuo del sistema sin interrumpir las acciones de los usuarios.

A partir de estos requisitos, se podrá avanzar en el diseño e implementación de una solución que satisfaga los objetivos planteados en el capítulo primero.

Estos requisitos los hemos separado en dos categorías diferentes: los requisitos funcionales, que describen las capacidades específicas que debe ofrecer el entorno para cumplir con los objetivos del proyecto, y los requisitos no funcionales, que detallan las características técnicas y de rendimiento necesarias para asegurar su estabilidad y eficiencia.

A continuación, presentamos los requisitos funcionales que recopilamos, centrados en definir las acciones y comportamientos esperados del entorno y de la herramienta dentro del marco de un hackatón de ciberseguridad. Estos requisitos permitirán validar que el sistema cumple su propósito de manera efectiva durante situaciones reales de uso, donde el tiempo, la estabilidad y la capacidad de respuesta son factores críticos:

- **Compatibilidad con Linux:** El entorno está planeado para que utilice el sistema operativo Ubuntu 24.04, asegurando que no afecte negativamente al rendimiento y que concuerde con el sistema operativo existente en los laboratorios de la facultad.
- **Diseño de una herramienta de auditoría:** Esta herramienta nos permitirá, al ejecutarla, obtener información acerca de todo el sistema operativo, incluyendo los siguientes módulos:

- *Información del sistema:* información general del sistema operativo, datos sobre el kernel y actualizaciones.
- *Red y comunicaciones:* configuraciones de la red actual y sus diferentes protocolos de seguridad.
- *Seguridad del sistema y accesos:* permisos de usuarios y grupos, usuario root y sudo, seguridad de aplicaciones y permisos de archivos.
- *Archivos y directorios:* información sobre el directorio de inicio y la seguridad de los diferentes archivos y directorios.
- *Procesos de memoria y actividad:* análisis de la información de la memoria y sus procesos, así como de los logs.
- *Backup y recuperación:* análisis de las copias de seguridad del sistema.
- *Tests*: pruebas Debian y análisis de resultados.
- *Protección y detección de amenazas:* búsqueda de antivirus instalados y diferentes rootkits.

- **Herramienta de monitoreo en tiempo real:** Desarrollo de una herramienta, capaz de detectar y notificar eventos del sistema de manera inmediata, incluyendo:
  - Accesos al sistema (inicios de sesión y fallos de autenticación).
  - Intentos de escalación de privilegios (uso de sudo, intentos de cambiar a root).
  - Modificaciones en archivos críticos (cambios en /etc/passwd, /etc/shadow, /var/log/auth.log).
  - Creación o eliminación de procesos sospechosos.
  - Conexiones de red entrantes y salientes.
- **Herramienta de defensa activa:** Implementación de un daemon de respuesta inmediata que, además de detectar amenazas, aplica contramedidas automáticas y registra cada acción.
- **Alertas sin interrupciones:** Es esencial que la herramienta de monitoreo registre eventos sin detener o interferir con los procesos en ejecución en el sistema. En lugar de bloquear acciones, debe registrar intentos sospechosos y reportarlos sin afectar la actividad del usuario.

- **Interfaz efectiva/resumida/seleccionada:** La herramienta debe presentar la información de manera clara y concisa, resaltando los datos críticos.

Finalmente, para garantizar no solo la funcionalidad del entorno, sino también que sea eficiente, fiable y seguro durante el desarrollo de un hackatón, se han definido los siguientes requisitos no funcionales. Estos requisitos establecen los estándares de calidad técnica que debe cumplir la herramienta, incluyendo aspectos relacionados con el rendimiento, la escalabilidad, la seguridad del sistema y la facilidad de mantenimiento.

- **Eficiencia:** La herramienta debe utilizar una cantidad mínima de CPU y RAM para evitar cualquier impacto negativo en el rendimiento del sistema.
- **Seguridad:** Solo los usuarios autorizados deben tener acceso a los reportes generados por la herramienta, garantizando la protección de la información.
- **Escalabilidad:** La herramienta debe ser capaz de monitorear múltiples eventos simultáneamente sin pérdida de rendimiento o precisión.
- **Integración con GitHub:** Debe permitir el control de versiones y facilitar la colaboración en el desarrollo mediante la integración con GitHub.

Capítulo 2 - Estado del Arte

En este capítulo presentamos un estudio detallado de lo que significa qué es un hackatón y lo diferentes modelos de hackáthones que hemos investigado para el desarrollo de este trabajo

La palabra "hackatón" proviene de la unión de dos palabras que, unidas, representan la definición de esta: "hacker" (soluciones tecnológicas) y "maratón" (una competencia larga). Por consiguiente, se trata de una competición en la que los participantes se organizan en grupos pequeños, para trabajar conjuntamente en soluciones para sus proyectos. Es decir, "un hackatón se apoya en la inteligencia colectiva y la colaboración" [2]. Además, según la Fundeo "como equivalente en castellano se emplea con mucha frecuencia la adaptación hackatón, con tilde en la o y sin la h final", que es la terminología que utilizaremos en todo este trabajo.

El origen registrado de los hackatones se remonta al año 1999 en Canadá, como un evento organizado por un grupo de desarrolladores de *OpenBSD* [3] en el cual establecieron un hito en la industria del software. Poco después, *Sun Microsystems* [4] celebró un evento similar, donde se desafió a los desarrolladores a crear un programa para un nuevo dispositivo.

Los impresionantes resultados producidos en estos eventos en tan poco tiempo no tardaron en propagarse, creciendo en popularidad desde entonces, hasta llegar a ser organizados con frecuencia por muchas de las empresas más grandes del mundo. Esta fama logró incluso trasladar el concepto de hackatón a muchas otras áreas como son la educación, la medicina y otros tipos de negocios, entre otros campos.

Entre los beneficios que aportan la celebración de estos eventos, destacamos el aprendizaje colaborativo y la posibilidad de encontrar enfoques diversos, ya que mejora los resultados obtenidos al obtener diferentes tipos de solución a un problema.

Para poder llevar a cabo la celebración de un Hackatón, lo primero es tener claro los objetivos a cumplir y planear una buena organización.

Los organizadores son los encargados de exponer el reto objetivo de este evento, aunque por lo general, se suele dar libertad a las ideas que van surgiendo de los participantes no estableciendo márgenes muy estrictos.

Los tiempos también deberían estar claramente marcados. La duración de estos tipos de eventos suele ser de uno o dos días, aunque es posible extender esta duración, dependiendo de la complejidad del objetivo, por lo que es importante planificarlo adecuadamente.

Luego, es muy importante contar con cierto número de participantes. Generalmente, estos deberían de tener conocimientos sobre el tema a tratar, dependiendo de la temática y objetivos. Estos eventos suelen requerir bastantes participantes (entre 20 y 50 según nuestra referencia) y creemos que sería conveniente hacer una buena selección de los participantes para poder asegurar algo de éxito al proyecto. Por ejemplo, no sería conveniente incluir al evento participantes sin compromiso ni actitud proactiva para colaborar en el proyecto.

Además de todo esto, ofrecer premios suele ser un buen incentivo para motivar a los participantes a esforzarse al máximo en el proyecto. Durante la investigación sobre este tipo de eventos, hemos llegado a encontrar grandes premios en metálico e incluso invitaciones con los gastos pagados a localizaciones como Bruselas, para participar de nuevo en otro Hackathon organizado por la misma empresa.

Finalmente, es muy importante tener en cuenta la organización de los recursos en caso de que el evento se celebre presencialmente. El espacio donde se celebra el evento; el acceso a comida con flexibilidad de dietas ya sea pagada por el evento o no; y la disposición de ordenadores o materiales para poder llevar a cabo las soluciones, son unos factores a tener en cuenta.

Un hackatón típico arranca con una sesión de bienvenida en la se exponen los objetivos, las actividades y la agenda, adaptando el formato a si el evento es presencial o virtual. A continuación, se forman equipos y se procede a trabajar, cada grupo en su estación, desarrollando los primeros prototipos, recibiendo retroalimentación de los supervisores y ajustando su enfoque hasta dar con una solución viable.

Al llegar al final del tiempo asignado, los equipos presentan sus avances ante un jurado que evalúa los prototipos y otorga premios, llegando incluso en ocasiones a invitar a colaborar en la puesta en marcha de las propuestas más prometedoras. El cierre del hackatón incluye un discurso de conclusión que resume las principales aportaciones y aprendizajes, que suele ir seguido de charlas informales o sesiones de networking que permiten ampliar la experiencia y fomentar futuras colaboraciones.

Además de hablar de los hackatones de ciberseguridad, otro elemento clave de este trabajo son las herramientas, más concretamente las herramientas enfocadas en la ciberseguridad.

En el ámbito de la informática, las herramientas juegan un papel crucial en la gestión, operación y optimización de sistemas. Estas herramientas pueden ser desde simples utilidades de línea de comandos hasta complejas aplicaciones de software diseñadas para automatizar tareas y mejorar la eficiencia operativa.

Las herramientas permiten a los usuarios realizar diferentes tareas de forma más eficiente. Por ejemplo, en Linux, utilidades como *cat* y *nano* facilitan la lectura y modificación de texto, mientras que *systemctl* gestiona servicios del sistema de manera estructurada.

En el ámbito de la ciberseguridad, estas herramientas no solo mejoran la productividad, sino que también permiten implementar estrategias de protección avanzadas de una manera más accesible. Por ello, para poder llevar a cabo nuestro cometido de crear un entorno para celebrar hackatones de ciberseguridad, hemos desarrollado unas herramientas personalizadas que lo permiten.

2.1 Hackatones de ciberseguridad

Cuando hablamos de un hackatón de ciberseguridad, nos estamos refiriendo a que el objetivo primordial del evento está relacionado con resolver desafíos de seguridad informática en tiempo real. En estas competiciones, equipos multidisciplinares trabajan contra reloj para identificar y explotar vulnerabilidades en un entorno controlado, aplicando técnicas de pentesting, hardening y análisis forense. Además, mientras resuelven los retos, afianzan sus capacidades relacionadas con la ciberseguridad, el

trabajo en equipo y la capacidad de gestión de proyectos, habilidades muy útiles para los participantes.

Para facilitar este proceso, los organizadores suelen poner a disposición de los participantes máquinas virtuales o contenedores preconfigurados con acceso root limitado y repositorios de código fuente, entre otros recursos. De este modo, se simulan escenarios reales de ataque y defensa sin comprometer la infraestructura del laboratorio.

Un formato especialmente popular en hackatones de ciberseguridad es la competición *Capture The Flag* (CTF) [5], en la que los organizadores plantean una serie de retos técnicos (desde explotación web y criptografía hasta ingeniería inversa) cuyo objetivo es encontrar los "flags" ocultos en programas o sistemas deliberadamente vulnerables. Entonces los participantes obtienen puntos al capturar cada flag, normalmente siguiendo un esquema de puntuación y una tabla de clasificación con resultados en tiempo real, que fomenta la competitividad entre los participantes. Este formato, constituye la modalidad más extendida en competiciones de seguridad hoy en día, tanto en entornos académicos como profesionales.

Entre los eventos de referencia que han ayudado a consolidar este formato destacan el *OpenBSD* [3] Crypto Hackatón de 1999, considerado uno de los primeros, tal y como vimos en la introducción; el *Pwn2Own* [6], lanzado en 2007 durante la conferencia *CanSecWest*, que premia económicamente el descubrimiento y divulgación responsable de vulnerabilidades en software de uso masivo; y el *DEF CON* [7] CTF, que desde principios de los años 2000 se ha llegado a posicionar como la "gran final" de las competiciones de captura de banderas, reuniendo anualmente a los equipos más expertos. Más recientemente, iniciativas como *Google CTF* [8] han ampliado el alcance de estos eventos a un público mucho mayor, combinando fases online y finales presenciales, y ofreciendo retos diseñados tanto para iniciados en el mundo de la ciberseguridad, como para expertos.

Cada uno de estos hackatones ha contribuido a establecer las mejores prácticas en organización, evaluación de resultados y divulgación de conocimiento, convirtiéndose así en auténticos laboratorios de innovación donde las empresas y universidades

pueden formarse, detectar tendencias emergentes en ciberamenazas y colaborar en soluciones de futuro.

2.1.1 Universidad Complutense de Madrid

A colación de todo lo anterior, conviene introducir las consideraciones básicas sobre la celebración de los hackatones de ciberseguridad en el ámbito universitario. Concretamente nos centraremos en la Universidad Complutense de Madrid (en adelante, UCM) y más específicamente la Facultad de Informática.

La UCM ha celebrado previamente Hackatones de diversas temáticas, tanto presencialmente como telemáticamente, lo cual demuestra interés por parte de la universidad en llevar a cabo estos eventos. Sin embargo, hasta la fecha no se han celebrado ningún hackatón relacionado con la ciberseguridad. Investigando por la web de la UCM vemos que las temáticas han estado relacionadas con la salud, el emprendimiento y las causas sociales.

Al analizar estos hackatones celebrados con anterioridad, vemos ciertos elementos que se podrían replicar para poder llevar a cabo estos tipos de eventos. Lo primero que vemos primordial sería que la UCM estuviera de acuerdo en celebrar un evento como este. Una vez que la universidad concede el visto bueno, sería conveniente encontrar unos patrocinadores que aporten financiación al evento y mejore los fondos con los que contaría la organización. En anteriores eventos hemos observado que se ha recibido patrocinio del Consejo Social de la UCM o del banco Santander Universidades, por lo que se podría aprovechar estas relaciones para llevar a cabo este tipo de hackatón.

Otro tema a tener en cuenta sería valorar si con la comunidad de estudiantes de la UCM sería suficiente para llevar a cabo el hackatón o de si sería necesario abrir las inscripciones a gente no adscrita a la universidad.

Finalmente, de cara a la celebración sería importante valorar el lugar de la celebración del evento. Tal y como estamos planteando nuestra herramienta, sería conveniente celebrar este evento en los laboratorios de la facultad de informática de la UCM, pero haría falta configurar previamente el laboratorio. Además, habría que proporcionar a los participantes una cuenta para acceder a los equipos de manera temporal.

En conclusión, estos serían los pasos a tener en cuenta en el caso de que se quisiera llevar a cabo un evento como este en la UCM. En el caso de que, tras este proyecto se logre con éxito los objetivos establecidos y tuviéramos el visto bueno de nuestro tutor, llevar a cabo un hackatón con nuestro entorno, podría ser el siguiente paso de este TFG.

2.2 Otros Tipos de Hackatones

Tal y como mencionamos en la introducción, existen diversas modalidades de hackatones, cada una orientada a objetivos y públicos específicos y distintos. Hemos decidido profundizar sobre los campos donde más son empleados los hackatones, para mejorar así nuestro entendimiento sobre ellos.

Los hackatones corporativos están organizados por empresas que buscan impulsar la creatividad interna y acelerar soluciones a retos de negocio. Por otra parte, los hackatones de propósito social reúnen a desarrolladores, ONGs y voluntarios para crear prototipos de soluciones orientados a problemas de salud, educación o medio ambiente, utilizando a menudo datos abiertos y fomentando alianzas con entidades públicas y privadas. En el ámbito académico destacan los hackatones estudiantiles, organizados por universidades como la UCM, que normalmente, con la ayuda y supervisión de profesores, desarrollan competencias prácticas y de gestión de proyectos. Existen también los hackatones temáticos, especializados en sectores o tecnologías concretas como, por ejemplo, blockchain, inteligencia artificial o movilidad urbana. Existen también los hackatones virtuales, los cuales ofrecen una experiencia completamente en línea, superando barreras geográficas y permitiendo la participación mediante plataformas de colaboración, repositorios Git y/o entornos en la nube sin instalaciones locales.

En definitiva, los hackatones abarcan un espectro muy amplio de usos y su creciente diversidad permite diseñar eventos a medida de cualquier objetivo como hemos ido viendo a lo largo de este capítulo. Fomentar la creatividad interna de una empresa, impulsar soluciones de impacto social, consolidar habilidades técnicas en estudiantes o explorar tecnologías punteras, por ejemplo, son solo una pequeña cantidad de los beneficios que puede aportar una empresa. Esta variedad no solo enriquece el ecosistema de innovación, sino que ofrece a los organizadores la flexibilidad necesaria

para alinear la temática, el formato y los recursos con las metas concretas de cada proyecto.

Por ello, comprender las características y ventajas de cada tipo de hackatón resulta clave para diseñar experiencias formativas y competitivas realmente efectivas, tal y como se pretende en nuestro TFG al adaptar el modelo de ciberseguridad a las necesidades de la UCM.

2.3 Herramientas en Ciberseguridad

Las herramientas en el ámbito de la **ciberseguridad** desempeñan un papel fundamental en la protección de la infraestructura digital. Permiten a las organizaciones prevenir, detectar y responder a amenazas, contribuyendo a garantizar los principios de seguridad informática CIA (*confidencialidad, integridad y disponibilidad*) de la información.

Este tipo de herramientas trabajan en conjunto con políticas de seguridad y procedimientos operativos para crear un ecosistema robusto frente a ataques cibernéticos. Su uso adecuado hace que se reduzca el riesgo de incidentes y nos permite reaccionar de manera rápida y efectiva ante amenazas en evolución.

Estas pueden clasificarse en dos grandes categorías según su cometido, las herramientas defensivas y las ofensivas. Las *herramientas defensivas* [9], están diseñadas para proteger los sistemas y redes de una organización contra amenazas. Encontramos herramientas como firewalls, sistemas de detección de intrusos (IDS [10]), antivirus y antimalware. Estas detectan patrones sospechosos en el tráfico de red o archivos ejecutables, bloqueando o alertando sobre posibles ataques.

Por otro lado, las *herramientas ofensivas* [11], se utilizan para identificar vulnerabilidades antes de que los atacantes las exploten. Entre ellas encontramos las herramientas de pentesting como *Metasploit* [12], *frameworks* de explotación de redes y herramientas de *fuzzing*, las cuales simulan ataques controlados para evaluar la seguridad de sistemas, permitiendo a los administradores reforzar sus defensas.

Estas dos categorías principales abarcan los **objetivos principales** de este tipo de herramientas. Los cuales son la detección de vulnerabilidades y amenazas, la

protección de sistemas y redes, la respuesta ante incidentes y el análisis forense digital, que son también esenciales para gestionar y mitigar los efectos de un ciberataque.

2.3.1 Clasificación de las Herramientas de Ciberseguridad

Los usos de las herramientas en el ámbito son extensos, tal y como hemos podido observar, por ello, hemos identificado una serie de categorías generales en las cuales hemos incluido cada uno de los ámbitos en los que se agrupan las herramientas.

Herramientas de Protección y Seguridad:

Las herramientas de protección y seguridad son clave para establecer mecanismos de defensa que prevengan ataques cibernéticos. Entre ellas hemos destacado las siguientes:

- **Antivirus y Antimalware:** Detectan, previenen y eliminan software malicioso como virus, gusanos, troyanos y *ransomware*. Utilizan técnicas como detección por firmas, heurística y s*andboxing* para identificar amenazas desconocidas.
- **Cortafuegos (Firewalls):** Actúan como una barrera entre redes internas seguras y redes externas no confiables, filtrando el tráfico entrante y saliente basado en reglas de seguridad predefinidas
- **Sistemas de Prevención de Intrusos (IPS):** Monitorean la red y los sistemas para detectar y bloquear actividades maliciosas en la red. Como, por ejemplo: Suricata [13] o Snort [14] en modo IPS.
- **Control de Acceso a la Red (NAC):** Gestionan y controlan el acceso a la red, asegurando que solo los dispositivos autorizados puedan conectarse. Como, por ejemplo: Cisco ISE [15] o PacketFence [16].

Herramientas de Monitoreo y Análisis:

Las herramientas de monitoreo y análisis son cruciales para la detección de actividad sospechosa y la auditoría de sistemas. Estas herramientas consisten en las siguientes categorías:

- **Sistemas de Detección de Intrusos (IDS):** Supervisan el tráfico de la red en busca de patrones de ataque o comportamientos anómalos. Como, por ejemplo: Snort o Zeek [17].
- **Plataformas de Monitoreo de Red:** Permiten analizar en tiempo real lo que sucede en la infraestructura de red. Como, por ejemplo: Wireshark [18] (inspección de paquetes) o Nagios [19] (monitoreo de sistemas).
- **Análisis de Logs:** Recopilan y analizan registros de eventos para detectar actividades inusuales o no autorizadas. Como, por ejemplo: ELK Stack [20] (Elasticsearch, Logstash, Kibana), Wazuh [21] o Splunk [22].
- **Sistemas Señuelo (Honeypots)**: Simulan servicios o sistemas vulnerables para atraer a posibles atacantes y registrar sus acciones en un entorno controlado. Estos señuelos permiten detectar accesos no autorizados, estudiar técnicas de ataque y generar alertas sin comprometer activos reales. Algunos ejemplos conocidos son *Dionaea* [23] (malware y exploits), *Honeyd* [24] (emulación de redes) o *Canarytokens* [25] (archivos trampa que activan alertas al ser accedidos).

Herramientas de Pruebas y Simulación de Ataques:

Las herramientas de pruebas y simulación de ataques se utilizan para evaluar la seguridad mediante tests controlados. Estas herramientas incluyen:

- **Pruebas de Penetración (Pentesting):** Simulan ataques reales para identificar y explotar vulnerabilidades en el sistema. Como, por ejemplo: Metasploit [12], Burp Suite [26] o Nmap [27].
- **Simulación de Brechas y Ataques (BAS):** Evalúan continuamente la seguridad simulando ataques automatizados. Como, por ejemplo: Cymulate [28] o SafeBreach [29].

Herramientas de Respuesta y Análisis Forense:

Las herramientas de respuesta y análisis forense son esenciales para la investigación de incidentes y la recuperación de datos. Estas herramientas están divididas en las siguientes:

- **Análisis Forense Digital (DFIR):** Permiten la adquisición y análisis de evidencia digital para investigar incidentes de seguridad. Como, por ejemplo: Autopsy [30], Volatility [31] (análisis de memoria RAM).
- **Recuperación de Datos:** Restauran información eliminada o dañada durante un ciberataque. Como, por ejemplo: TestDisk [32] o Recuva [33].
- **Registro y Reportes:** Herramientas que generan informes detallados sobre incidentes de seguridad y ayudan a preservar la cadena de custodia de las pruebas. Como, por ejemplo: Splunk [34], Graylog [35], Wazuh [21].

2.4 Herramientas que hemos utilizado en el diseño del Hackatón

Para realizar este Trabajo de Fin de Grado, hemos partido exclusivamente de los objetivos y el modo en el que operan las herramientas existentes. Todo el código de las herramientas desarrolladas ha sido implementado por nosotros desde cero. En lugar de emplear soluciones de monitorización ya creadas, diseñamos nuestras propias herramientas que unifican las funciones habituales de un IDS y la gestión de acceso a la red, así como la recolección y el análisis de logs para detectar actividades sospechosas.

Con la MonitoringTool adaptamos internamente las capacidades de los honeypots (generación de señuelos, alertas automáticas y bloqueo de accesos), mientras que AuditTool, además de nuestras propias comprobaciones, incorpora chkrootkit y ClamAV para realizar análisis de rootkits y malware, y produce así informes comparativos muy completos del estado del sistema antes y después del hackatón, seleccionando únicamente la información que hemos considerado esencial para evaluar la integridad de los equipos. De este modo, garantizamos un entorno autónomo y coherente con los requisitos de un evento de ciberseguridad sin depender de software de terceros.

Por último, la herramienta Defense se basa en un sistema de detección de eventos de log en áreas críticas del sistema, definidas expresamente en el código. Cuando detecta actividad sospechosa, envía automáticamente una alerta por correo electrónico al responsable, replicando el funcionamiento de un IDS.

Capítulo 3 - Entorno y Herramientas: Ubuntu 24.04 y Python para ciberseguridad

En este capítulo, se contextualiza respecto al sistema operativo en el que se basa el entorno creado en este trabajo y en el lenguaje de programación de las herramientas creadas.

Comprender el sistema operativo y el lenguaje ha sido fundamental para la realización del presente trabajo.

3.1 Sistema Operativo: Ubuntu 24.04

3.1.1 Elección de Ubuntu

Nos hemos decantado por Ubuntu 24.04 LTS no solo ser el sistema operativo en uso en los laboratorios de la Facultad de Informática, sino también por las ventajas que ofrece GNU/Linux [36] en este tipo de eventos frente a sistemas operativos privativos. Al ser de código abierto, Linux nos permite auditar y personalizar las configuraciones de seguridad sin restricciones, además de permitirnos reproducir el entorno mediante máquinas virtuales sin costes de licencia adicionales. Además, la mayoría de las herramientas de pentesting y análisis forense tienen su origen o mejor soporte en Linux, lo que nos facilita su instalación y uso durante el desarrollo del entorno. Por último, el robusto sistema de permisos de Unix y la disponibilidad de distribuciones ligeras garantizan un rendimiento óptimo y una rápida recuperación ante fallos, lo cual es muy importante en hackatones intensivos de 24 o 48 horas. En nuestro caso, usar la versión ya instalada en el laboratorio nos permitió trabajar sobre un entorno familiar, minimizando la dificultad de trabajar en la infraestructura existente. Ubuntu 24.04 LTS [37], lanzado en 2024, es una versión con soporte durante cinco años, que nos proporciona un sistema al orden del día y respaldado a largo plazo, que lo hace ideal para un proyecto como el nuestro.

3.1.2 Características y arquitectura del sistema.

Ubuntu forma parte de la familia de distribuciones GNU/Linux, basado en el kernel Linux [38]. El kernel es el componente central que gestiona los recursos del hardware y las

comunicaciones entre procesos, consiguiendo así que las aplicaciones se ejecuten de manera controlada. Internamente, el kernel de Linux emplea un planificador que distribuye el tiempo de CPU entre procesos de forma equilibrada, manteniendo el sistema funcionando hasta cuando está sometido a altas cargas de trabajo. Esta característica es fundamental durante la celebración de hackatones, donde múltiples procesos (y eventualmente contenedores) intentan acceder a los recursos. El sistema de archivos de Ubuntu es jerárquico y organiza los datos de forma estructurada, por ejemplo, /etc alberga configuraciones del sistema, /var contiene registros (*logs*) y datos variables, /home agrupa los directorios de usuarios y /tmp almacena archivos temporales. Entre todos los archivos que contiene el sistema, es importante asegurar la integridad de ficheros sensibles como los de /etc, ya que cualquier modificación no autorizada en la configuración, podría comprometer el funcionamiento del entorno y de las herramientas del hackatón, tal y como estaría planeado.

Para la gestión de memoria, el kernel aprovecha la memoria virtual mediante un área de intercambio (swap) que evita posibles errores por falta de RAM al mover los datos menos usados al disco. No obstante, un uso excesivo de swap puede ralentizar el sistema. Para optimizar el rendimiento debido a la alta carga del hackatón, se podría ajustar parámetros como el nivel de *swappiness* [39] , para que el sistema priorizase el uso de RAM y solo recurriera al intercambio cuando fuese realmente necesario. Con estos ajustes, mantendríamos un entorno fluido incluso cuando múltiples herramientas de auditoría y monitorización consuman memoria simultáneamente.

3.1.3 Gestión de usuarios y seguridad del sistema.

Ubuntu maneja un modelo de seguridad basándose en usuarios con distintos niveles de privilegio. El superusuario root posee control total, mientras que los usuarios normales solo tienen permisos limitados. En lo que se refiere a las tareas de administración del sistema (como instalar un archivo, por ejemplo) existen permisos temporales que pueden ser concedidos a través de la función sudo, de forma que ciertos usuarios puedan ejecutar comandos privilegiados con sudo sin exponer directamente la cuenta root. El sistema almacena las credenciales de forma segura como los hashes de contraseñas en el directorio /etc/shadow, utilizando algoritmos criptográficos (como SHA-512),

protegiendo así las credenciales contra accesos no autorizados mediante fuerza bruta. Adicionalmente soporta métodos de autenticación flexibles mediante PAM [40] (Pluggable Authentication Modules), o incluso esquemas de múltiple factor (MFA). En nuestro entorno de hackatón, sin embargo, operamos principalmente con cuentas locales de laboratorio y contraseñas seguras, considerando suficiente esta capa dado que el evento se desarrolla en un entorno controlado.

Para mejorar el control de acceso del sistema operativo, Ubuntu incluye de serie AppArmor [41] como mecanismo MAC (*Mandatory Access Control*). AppArmor confina la ejecución de procesos mediante *perfiles de seguridad* que definen a qué archivos y a qué recursos puede acceder cada aplicación, reduciendo así el impacto de posibles vulnerabilidades en algún software. De este modo, incluso si durante el hackatón algún servicio o herramienta es comprometido por los participantes, AppArmor limita el alcance del daño al restringir qué tipo de acciones puede llevar a cabo el proceso comprometido. Ubuntu tiene también las Listas de Control de Acceso (ACL) que define para los sistemas de archivos, permisos más granulares que los tradicionales de UNIX. Sin embargo, en nuestra implementación principalmente hemos empleado los permisos estándar.

En el plano de la red, protegemos el sistema con un firewall. Ubuntu dispone de la herramienta UFW (*Uncomplicated Firewall*), que es una herramienta que simplifica el uso de las mismas funcionalidades que *iptables*. Hemos configurado UFW para restringir el tráfico no deseado, permitiendo solo las comunicaciones necesarias para el desarrollo del hackatón. Por ejemplo, se pueden bloquear conexiones entrantes y solo habilitar puertos relevantes para las prácticas definidas, evitando así accesos no autorizados a los equipos de los participantes. Para necesidades más avanzadas, *iptables* ofrece un mayor control del tráfico, aunque su uso requiere mayor conocimiento. En nuestro caso UFW resultó suficiente para aplicar una política de "denegar todo por defecto" y abrir excepciones puntuales, garantizando una protección de red adecuada con mínima complejidad.

3.1.4 Administración y monitorización del sistema.

Uno de los motivos por los que Ubuntu destaca es por la abundancia de herramientas de administración que aporta de serie, lo cual nos ha permitido integrar nuestras soluciones con el sistema de forma fluida. Ubuntu utiliza *systemd* como sistema de inicialización y gestión de servicios. Esto nos ha facilitado el despliegue de nuestras herramientas de seguridad como servicios del sistema, de modo que se inician automáticamente al arrancar la máquina y se mantienen en ejecución monitorizando el entorno durante todo el hackatón. Mediante comandos de *systemctl* hemos podido gestionar estos *daemons* e incluso configurar reinicios automáticos en caso de fallo, aumentando la robustez de nuestras soluciones.

Para la supervisión en tiempo real del estado del sistema, disponemos de utilidades como top/htop o Glances, que nos permiten observar el uso de CPU, memoria y otros recursos. Sin embargo, en lugar de depender solo de herramientas interactivas, nuestro enfoque ha sido desarrollar scripts en Python que recojan métricas e informen de eventos importantes automáticamente. En este sentido, resulta fundamental el acceso a los logs del sistema. Ubuntu registra eventos y mensajes en el journal de systemd (vía journald) y en archivos de texto plano bajo /var/log/ (administrados por rsyslog). Para nuestro proyecto, hemos aprovechado ambas fuentes, por un lado, el comando *journalctl* nos permite consultar el historial de eventos del sistema en formato binario con gran detalle, y por el otro lado, archivos como /var/log/auth.log o /var/log/syslog nos sirven para detectar intentos de acceso o comportamientos anómalos. También activamos el servicio auditd, un demonio [42] de auditoría de Linux, para que registre eventos de seguridad de alto valor, como modificaciones en archivos críticos, cambios de permisos o ejecuciones de comandos clave. En este TFG, nuestras herramientas procesan estos registros en tiempo real para generar alertas automáticas e incluso comparan el estado inicial y final de cada máquina al concluir el hackatón, identificando cambios significativos provocados durante las pruebas de intrusión. De esta manera podemos evaluar qué alteraciones sufrió el sistema (nuevos usuarios, puertos abiertos, servicios caídos, etc.) y restaurarlo o investigarlas según corresponda.

3.2 Lenguaje de programación: Python

Python no solo es un lenguaje ampliamente empleado, sino que también ha ganado un papel crucial en la automatización de tareas, pruebas de penetración (*pentesting*), análisis de vulnerabilidades y criptografía.

En los últimos años, su popularidad en el sector de la seguridad ha crecido exponencialmente, convirtiéndolo en una de las herramientas esenciales para investigadores y analistas. El éxito de Python en ciberseguridad no solo se debe a la versatilidad y potencia que nos ofrece, sino también se debe a una serie de características que lo convierten en una opción preferida entre los profesionales del sector. Estas características facilitan la creación de herramientas de seguridad, optimizar los procesos de análisis y mejorar la detección y mitigación de amenazas.

3.2.1 Características principales de Python

- **Facilidad de aprendizaje e implementación:** Su sintaxis minimalista permite traducir rápidamente ideas a código, reduciendo la curva de entrada y acelerando el desarrollo.
- **Facilita la depuración y el mantenimiento del código:** Con la estructura clara y ordenada que mantiene, reduce los tiempos de desarrollo y facilita la identificación de errores y la colaboración entre programadores de distintos niveles de experiencia.
- **Código abierto y comunidad activa:** Al ser un proyecto de código abierto, existen miles de bibliotecas y recursos compartidos, con documentación y soporte continuo de la comunidad, que facilita su aprendizaje y la resolución de problemas.
- **Gestión automática de memoria:** Integra un sistema que administra la memoria sin intervención manual, previniendo fugas y mejorando la estabilidad y el rendimiento

Estas cuatro características generales explican diferentes usos de Python en diferentes campos, incluyendo al ámbito de la ciberseguridad, pero hay algunas más concretas

que explican por qué este lenguaje es el más usado en el ámbito de la seguridad informática.

Python es lo suficientemente potente para llevar a cabo una amplia variedad de tareas de ciberseguridad sin necesidad de recurrir a otros lenguajes de programación. Desde escaneo de redes y análisis de vulnerabilidades hasta detección de malware y automatización de pruebas de penetración. Además, la posibilidad de desarrollar scripts de manera rápida debido a la facilidad de código es una gran ventaja.

Finalmente, uno de los puntos más importantes y la explicación del uso de Python en ciberseguridad, es su amplia y extensa biblioteca. Python cuenta con una gran colección de módulos preexistentes, lo que permite a los profesionales de la ciberseguridad acceder rápidamente a herramientas y funciones necesarias para realizar tareas complejas.

3.2.2 Automatización con Python

Los equipos de seguridad enfrentan constantemente nuevos casos de uso que generan alertas e incidentes, los cuales deben ser gestionados y evaluados por analistas e ingenieros. A medida que los equipos y programas de seguridad maduran, la introducción de automatización personalizada se convierte en una prioridad.

Python permite a los profesionales de seguridad crear programas que optimicen las tareas diarias de protección de redes, detección de amenazas y análisis de incidentes.

Pasos para implementar un caso de uso de seguridad con Python para la preparación de un hackatón de ciberseguridad genérico:

1. **Definir el caso de uso**

   Acotar el objetivo concreto (p. ej. detección de accesos SUID, análisis de logs de monitorización) y los resultados esperados (alertas por correo, informe comparativo).

2. **Elegir fuentes de datos**

   Determinar qué registros o flujos generará la infraestructura del hackatón (auditd, journald, archivos de honeypot) y cómo capturarlos.

3. **Diseñar la lógica de seguridad**

   Traducir los requisitos en reglas y algoritmos automáticos (filtros de logs, patrones de tráfico, comprobaciones SUID) que Python ejecutará de forma continua.

4. **Configurar el Entorno:**

   Preparar un entorno virtual con Python y las bibliotecas necesarias (psutil, subprocess, smtplib…) y definir el despliegue como servicio systemd.

5. **Implementar y Validar**

   Desarrollar el script según la lógica definida, testearlo con datos reales del laboratorio y documentar su uso y parámetros.

Python permite automatizar prácticamente cualquier etapa de la seguridad informática gracias a su ecosistema de librerías especializadas. Entre los usos más habituales destacan el escaneo de red, el análisis de vulnerabilidades, el descifrado de contraseñas, el análisis de malware, la aplicación de criptografía, la detección de intrusiones, las transferencias seguras de archivos, la inteligencia de amenazas, el análisis y monitorización de logs, y la detección de ataques de fuerza bruta.

Con todo lo expuesto sobre las capacidades de Python en ciberseguridad, desde su rápida curva de aprendizaje y gestión automática de memoria hasta el uso de librerías especializadas para análisis de red, detección de intrusiones y criptografía, contamos ya con los cimientos técnicos y metodológicos necesarios para diseñar nuestras soluciones. Este recorrido nos ha permitido asentar buenas prácticas de desarrollo, definición de requisitos y despliegue de scripts como servicios del sistema, asegurando un entorno reproducible y fiable.

Capítulo 4 - Desarrollo de las Herramientas

Tras analizar los requisitos durante la fase de planificación del proyecto, se detectó la necesidad de desarrollar tres herramientas clave para construir el entorno en el que operarán los participantes del hackatón. Esta necesidad está en plena sintonía con los objetivos inicialmente planteados. En este capítulo, se presenta de manera detallada el propósito que cumple cada herramienta dentro del entorno seguro, el proceso seguido para su desarrollo, los comandos utilizados tanto para su creación como para su ejecución, los obstáculos que surgieron durante el desarrollo (incluyendo aproximaciones que no dieron resultado) y, finalmente, las métricas aplicadas para evaluar la complejidad técnica de cada uno de estos desarrollos.

En el proceso de desarrollo de este trabajo se decidió que era necesario desarrollar tres herramientas que se ocupasen de auditar el sistema a estudio, **Audit-tool**; monitorizar lo que está ocurriendo y ha ocurrido durante el hackatón, **Monitoring-Tool** y otra herramienta para tomar acciones en caso de que durante el desarrollo del hackatón preservara la integridad de los laboratorios, se produjeran pruebas o accesos no autorizados/deseados, **Defense-Tool**.

4.1 Audit-Tool

Audit-Tool fue desarrollada con el objetivo de realizar una auditoría integral del sistema, enfocándose en recopilar datos clave sobre la seguridad del host Linux e identificar configuraciones que pudieran resultar inseguras. A lo largo del desarrollo del proyecto, esta herramienta atravesó distintas etapas, evolucionando desde un prototipo básico hasta llegar a una versión final estable y funcional. En las próximas secciones, se expone en detalle cómo fue su proceso de desarrollo, los comandos utilizados para su construcción y puesta en marcha, los enfoques que no dieron resultado, así como las métricas empleadas para valorar su nivel de complejidad.

4.1.1 Proceso de desarrollo

Versión	Enfoque	Módulos	Mejoras clave
Prueba	Archivo único con toda la lógica	No modular, todo en un solo archivo	Primera implementación funcional
v1.0	Modular	10 módulos: sys_info, kernel, mem_process, users_groups_auth, debian_tests, boot_services, network, file_permissions, home_directories, storage_device	Separación en módulos; ejecución independiente por secciones
v1.1	Modular ampliado	10 nuevos módulos: logs, app_security, advanced_network_security, service_accounts, containers_security, updates, security_policies, sudo, storage_services, malware_protection, backup	Cobertura ampliada del sistema; mayor granularidad
v2.0	Modular estructurado por categorías	Reorganización de módulos en 9 categorías temáticas	Organización lógica, mejoras en sudo, claridad y escalabilidad

Tabla 1

El desarrollo de Audit-Tool comenzó con la creación de un repositorio Git local para gestionar el control de versiones. Se inicializó con git init en un nuevo directorio del proyecto, configurando la rama principal bajo el nombre convencional de main, y añadiendo un archivo README básico (git add README.md && git commit -m "Inicializar repositorio Audit-Tool"). A partir de ese punto, se definió una estructura simple de una sola rama, realizando commits conforme avanzaba el trabajo. Esto permitió mantener un seguimiento claro y ordenado de la evolución de la herramienta a lo largo del tiempo.

Inicialmente, Audit-Tool fue concebida como un único archivo en Python que contenía toda la lógica del programa. Ese script principal (Audit-Tool.py) integraba diversas funciones, cada una encargada de revisar un aspecto específico de la seguridad del

sistema. Por ejemplo, una función system_info() recuperaba información básica del sistema operativo; otras verificaban la configuración del kernel, revisaban permisos en archivos críticos, entre otras tareas. Esta aproximación permitió lograr una primera versión funcional de forma rápida. Sin embargo, con el tiempo se evidenciaron importantes limitaciones: el código resultaba difícil de mantener y cualquier error en una sola función podía comprometer toda la ejecución, lo que afectaba la estabilidad general de la auditoría. Esta fragilidad motivó una revisión profunda del diseño.

A partir de la versión v1.0, se adoptó una arquitectura modular. Se desarrolló un archivo principal actuando como lanzador, responsable de ejecutar distintos módulos especializados, cada uno alojado en archivos independientes dentro del paquete modules/. En esta versión, la herramienta se dividió en 10 módulos básicos, cada uno centrado en un área específica de la seguridad del sistema. Entre ellos se incluían: sys_info, kernel, mem_process, users_groups_auth, debian_tests, boot_services, network, file_permissions, home_directories y storage_device. Cada módulo contaba con funciones independientes que realizaban sus respectivas comprobaciones y devolvían resultados por separado. El archivo principal se encargaba de orquestar su ejecución secuencial y compilar un informe final en formato JSON. Este enfoque mejoró significativamente la robustez del sistema: un fallo en un módulo ya no afectaba al resto, y el desarrollo de cada componente podía abordarse de manera aislada, facilitando la depuración.

Con la versión v1.1 se amplió la cobertura mediante la incorporación de 10 módulos adicionales enfocados en aspectos más avanzados de seguridad. Algunos ejemplos fueron logs, para analizar eventos del sistema; app_security, centrado en configuraciones de aplicaciones instaladas; advanced_network_security, que revisaba reglas de firewall IPv6 y túneles VPN; o containers_security, para verificar entornos virtualizados. También se añadieron módulos como updates, security_policies, sudo, storage_services y malware_protection. Esta expansión llevó el total a 20 módulos, permitiendo una auditoría más detallada y completa, abarcando tanto configuraciones básicas como elementos más especializados que suelen pasarse por alto.

En la versión estable v2.0 se realizó una reorganización conceptual de los módulos para optimizar la claridad del código y su mantenimiento a largo plazo. Dado el número creciente de componentes, se optó por agruparlos temáticamente dentro del archivo principal mediante comentarios estructurados. Estas categorías internas permiten identificar rápidamente el propósito de cada módulo y facilitan la navegación por el código. Las categorías definidas fueron: Información del Sistema, Red y Comunicaciones, Seguridad del Sistema y Accesos, Archivos y Directorios, Procesos y Actividad, Contenedores, Backup, Tests, y Protección frente a Amenazas.

Uno de los cambios clave de esta versión fue la mejora del módulo sudo, al que se le añadieron nuevas capacidades. Dado su papel crítico en la gestión de privilegios, se implementaron verificaciones más profundas sobre el archivo sudoers, incluyendo detección de configuraciones inseguras como el uso de NOPASSWD y alertas sobre usuarios con permisos excesivos. Esta mejora fortaleció notablemente el valor práctico de la herramienta, especialmente dentro del contexto del hackatón, donde se anticipan intentos de escalada de privilegios como parte de las dinámicas defensivas u ofensivas.

Finalmente, se incorporó un documento complementario como Apéndice E, en el que se describe en detalle la arquitectura interna de Audit-Tool. Este anexo ofrece una explicación completa del diseño, funciones y lógica implementada en cada módulo, sirviendo como referencia técnica para desarrolladores y evaluadores del proyecto.

4.1.2 Estructura y Arquitectura de la herramienta

A continuación, se exponen las funcionalidades más relevantes y representativas de cada sección y módulo de la herramienta audit_tool, para ello se muestra el código de cada funcionalidad con el valor y la solución que aporta.

Como se puede ver en la Figura 1 a continuación, se incluye un diagrama visual de cómo está estructurada la herramienta. Dentro de la carpeta de los módulos, hemos creado una serie de secciones a modo de división del tipo de información del sistema que recopila, tal y como vimos en la sección anterior. Al final de cada módulo de cada sección se citarán las demás funcionalidades implementadas en herramienta final.

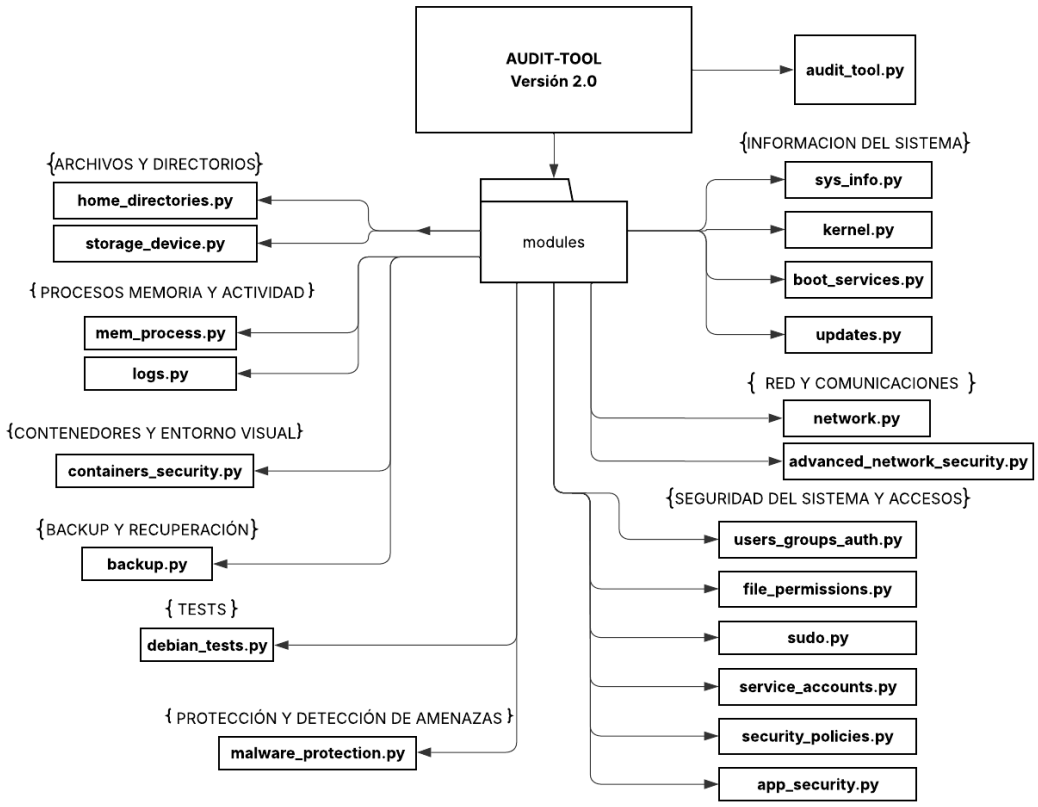

Figura 3: Estructura de la herramienta audit

4.1.2.1 Launcher

- ***audit_tool.py***

En primer lugar, chequeamos que el usuario ejecute el programa en modo root, a continuación, se procede a la carga independiente de los módulos, proporcionando así una mayor escalabilidad en el proyecto. Finalmente, como vemos en el código los resultados se guardarán en un JSON para tener una organización correcta de los datos.

```
    def run_selected_modules():
    results = {}
    #INFORMACIÓN DEL SISTEMA
    # 1 Ejecutar información del sistema primero
    print("[INFO] Ejecutando módulo: sys_info...\n")
    results["sys_info"]=importlib.import_module("modules.sys_info").run(
)
    # 2 Ejecutar auditoría del kernel
    print("\n[INFO] Ejecutando módulo: kernel...\n")
```

```
    results["kernel"] = importlib.import_module("modules.kernel").run()
```

Figura 4: Código audit_tool.py

Esta función es la más importante de la herramienta, ya que actúa como launcher y es clave para cargar cada módulo con importlib.import_module y ejecutar cada función run(). Almacena el resultado en un diccionario que después se importa como JSON. Su complejidad reside en un diseño estructurado de software y un control de errores para garantizar la correcta ejecución modular. Facilita la escalabilidad y hace que sea más sencillo añadir módulos nuevos sin comprometer los antiguos.

4.1.2.2 Información del sistema

- ***sys_info.py***

Obtiene información detallada del sistema y la devuelve en un diccionario. Permite obtener información importante, como el nombre del sistema operativo y su versión, la arquitectura del procesador, el nombre del equipo en la red, tiempo de actividad del sistema, etc. Esta función nos aporta información básica pero esencial del sistema.

```
    "program_version": "2.0.0",  # Versión de la herramienta
    "operating_system": platform.system(),
    "disk_usage": f"{psutil.disk_usage('/').percent}%",
    "user_running_script": os.getlogin(),
```

Figura 5: Código sys_info.py

- ***kernel.py***

El kernel es el elemento esencial para el funcionamiento de un sistema operativo, por ello elegimos este módulo como parte importante de la herramienta. Se encarga de diferentes tareas como procesos, gestiones de memoria, tareas de red, etc

Además de obtener la versión del kernel con "uname -r", una de las funciones más representativas y complejas es check_aslr().

```
    def check_aslr():
    """Verifica si ASLR (Address Space Layout Randomization) está
habilitado."""
    try:
        with open("/proc/sys/kernel/randomize_va_space") as f:
            aslr_value = f.read().strip()
```

```
            return "Habilitado" if aslr_value == "2" else "Deshabilitado"
      except Exception as e:
            return f"Error al verificar ASLR: {str(e)}"
```

Figura 6: Código kernel.py 1

Esta función es importante ya que ASLR [43] es una técnica de mitigación de ataques cuya activación es crítica. Requiere acceder a "/proc/sys/kernel/randomize_va_space", un archivo del espacio de usuario con valores del kernel, e interpretar su significado correctamente. Informa si el sistema aplica aleatoriedad al ubicar los procesos en memoria, dificultando ataques con exploits.

```
      def check_runlevel():
      """Verifica el nivel de ejecución predeterminado."""
      try:
            runlevel = subprocess.getoutput("runlevel")
            return f"Runlevel: {runlevel.split()[1]}" if runlevel else "No
se pudo obtener el runlevel"
```

Figura 7: Código kernel.py 2

El nivel de ejecución (runlevel) define el estado del sistema en Linux, como si está en modo multiusuario, con GUI, o en modo rescate. Ejecuta "runlevel", que devuelve un resultado en formato: "N 5", donde N es el nivel de ejecución anterior y 5 el nivel de ejecución actual. Conociendo el runlevel, sabremos cuantos servicios están activos, cuanto más alto sea el número más componentes se cargan y mayor es la superficie expuesta a vulnerabilidades, por ello es esencial la presencia de esta función, destacándose por encima de otras.

```
      kernel_version = get_kernel_version()
      aslr_status = check_aslr()
      kernel_parameters = check_kernel_parameters()
      loaded_kernel_modules = check_kernel_modules()
      kernel_hardening = check_kernel_hardening()
      runlevel = check_runlevel()
      cpu_support = check_cpu_support()
      kernel_type = check_kernel_type()
      io_scheduler = check_io_scheduler()
      kernel_updates = check_kernel_updates()
```

```
    core_dumps = check_core_dumps()
    reboot_needed = check_reboot_needed()
```

Figura 8: Código kernel.py 3

Estas son todas las funciones usadas en kernel.py, cada una de ellas obtiene información clave sobre el kernel y el sistema, destacando las funcionalidades explicadas anteriormente.

- ***boot_services.py***

```
    def check_service_manager():
    """Verifica qué gestor de servicios está en uso."""
    try:
        service_manager = subprocess.getoutput("ps -p 1 -o comm=")
```

Figura 9: Código boot_services.py 1

Determina qué gestor de servicios está en uso en el sistema, componente clave para la gestión de servicios del sistema como el arranque, reinicio, etc. Conocer si usa systemd, init u otro gestor es esencial para adaptar los análisis posteriores y garantizar la compatibilidad de comandos.

Ejecuta "ps -p 1 -o comm=" para obtener el nombre del proceso con PID 1 (el proceso padre de todos). Si devuelve systemd, significa que el sistema usa systemd. Si devuelve init, está usando SysVinit u otro sistema más antiguo.Si la salida está vacía, devuelve "Not Found".

```
    def check_password_protection():
    """Verifica si el sistema tiene protección por contraseña
configurada."""
    try:
        password_protection = subprocess.getoutput("grep -i password
/etc/grub.d/40_custom")
```

Figura 10: Código boot_services.py 2

Verifica si GRUB [44] tiene protección por contraseña configurada, el gestor de arranque GRUB, representa uno de los puntos más críticos y olvidados en la seguridad del sistema, por ello esta función es capaz de auditar este tipo de configuraciones. Es aconsejable

proteger el GRUB en diversas situaciones, cómo en entornos profesionales, para evitar accesos a nuestro ordenador con privilegios de root.

Usa "grep -i password /etc/grub.d/40_custom" para buscar configuraciones de contraseña en GRUB, si encuentra alguna referencia a "password", devuelve "Found", indicando que la protección por contraseña está configurada, si no encuentra nada, devuelve "Not Found".

```
    def run_systemd_analyze_security():
    """Ejecuta systemd-analyze security para verificar la seguridad de los
servicios."""
    try:
        security_analysis = subprocess.getoutput("systemd-analyze
security")
        return security_analysis
    except Exception as e:
        return f"Error al ejecutar systemd-analyze security: {str(e)}"
```

Figura 11: Código boot_services.py 3

Ejecuta "systemd-analyze security" para evaluar la seguridad de los servicios, es decir, los servicios definidos en systemd, evaluando así su configuración de seguridad devolviendo un informe con advertencias y puntuaciones. Las configuraciones revisadas se refieren al acceso del servicio al sistema de archivos, recursos del kernel y capacidades privilegiadas. Esta funcionalidad nos proporciona información sobre que servicios son más vulnerables, además de que nos sirve de evaluación para las dos herramientas que hemos desarrollado (monitoring_tool y audit_tool) ya que ambas se ejecutan como servicios.

Ejecuta "systemd-analyze security" para obtener información sobre la seguridad de los servicios en systemd, devolviendo la salida completa del comando.

- **updates.py**

```
    pending_updates = check_pending_updates()
    repository_configuration = check_repository_configuration()
    automatic_updates = check_automatic_updates()
```

Figura 12: Código updates.py 1

Las actualizaciones pendientes son uno de los vectores de ataque más frecuentes, por ello con este módulo se verifica si hay actualizaciones críticas pendientes, además de la configuración del repositorio. Destacando la primera función que ejecuta el comando "apt list --upgradable" obteniendo la lista de paquetes que pueden actualizarse, si hay paquetes marcados como "upgradable from", devuelve un mensaje indicando las actualizaciones pendientes, si no hay paquetes para actualizar, devuelve "No pending updates found.".

```
    def check_pending_updates():
    """Verifica si hay actualizaciones críticas pendientes."""
    try:
        updates = subprocess.getoutput("apt list --upgradable")
```

Figura 13: Códigos updates.py 2

4.1.2.3 Red y comunicaciones

- ***network.py***

```
    def check_ipv6_configuration():
    """Verifica la configuración de IPv6."""
    try:
        ipv6_status = subprocess.getoutput("sysctl
net.ipv6.conf.all.disable_ipv6")
        return "Enabled" if "0" in ipv6_status else "Disabled"
    except Exception as e:
        return f"Error al verificar configuración de IPv6: {str(e)}"
```

Figura 14: Código netwrok.py 1

Verifica si IPv6 [45] está habilitado o deshabilitado, identificando rápidamente una amenaza concreta y usual, la exposición por IPv6 no gestionada. La dirección IPv6 es un valor de 128 bits que identifica un dispositivo en un protocolo IPv6 (Internet Protocol Version 6). Este protocolo tiene diferentes ventajas, cómo un enrutamiento más eficiente gracias a tablas de enrutamiento más pequeñas y el agregado de prefijo, o, debido a cabeceras más optimizadas, un procesamiento de paquetes más simplificado.

Para ello, ejecuta "sysctl net.ipv6.conf.all.disable_ipv6", que devuelve 0 si IPv6 está habilitado y 1 si está deshabilitado.

```
    def check_dns_servers():
    """Verifica los servidores DNS configurados."""
    try:
        dns_servers = subprocess.getoutput("grep -i 'nameserver'
/etc/resolv.conf")
```

Figura 15: Código netwrok.py 2

Esta función obtiene los servidores DNS [46] configurados. Los servidores DNS (Domain Name Services) permiten acceder a sitios web utilizando los nombres de dominio y URL en lugar de IPs complejas, en las que las funciones de búsqueda DNS coinciden con su IP asignada. Si el sistema utiliza servidores DNS inseguros, un atacante puede capturar credenciales, realizar ataques MITM y redirigir conexiones, por todo esto es indispensable que esta configuración sea correcta.

En primer lugar se lee el archivo "/etc/resolv.conf", que contiene las direcciones de los servidores DNS configurados en el sistema, filtrando solo las líneas que contienen "nameserver" utilizando el comando "grep -i 'nameserver' /etc/resolv.conf". Si se encuentran servidores DNS, devuelve "OK" junto con la lista de servidores detectados (eliminando espacios en blanco con "strip())", si el archivo está vacío o no contiene servidores DNS, devuelve "Not Found".

```
    def get_listening_ports():
    """Obtiene los puertos de escucha (TCP/UDP)."""
    try:
        listening_ports = subprocess.getoutput("ss -tuln")
        return listening_ports if listening_ports else "Not Found"
    except Exception as e:
        return f"Error al obtener puertos de escucha: {str(e)}"
```

Figura 16: Código netwrok.py 3

Obtiene los puertos en escucha, que son los que esperan activamente recibir una conexión TCP o UDP [47]. Cada puerto abierto representa un servicio accesible desde la red, con la revisión de los puertos abiertos, identificaremos los vectores de ataque activos; siendo muy habitual encontrarnos puertos abiertos que no deberían de estar activos.

Usa "ss -tuln" para listar los puertos TCP y UDP que están en escucha.Si la salida contiene datos, los devuelve; de lo contrario, devuelve "Not Found".

```
    def check_promiscuous_interfaces():
    """Verifica si hay interfaces en modo promiscuo."""
    try:
          interfaces = subprocess.getoutput("ip link show | grep promisc")
```

Figura 17: Código network.py 1

Verifica si alguna interfaz de red está en modo promiscuo [48], este modo, es una configuración de los controladores de la interfaz de red que permiten al dispositivo capturar todo lo que está sucediendo en el tráfico de red, sin tener en cuenta el destinatario, una configuración extremadamente peligrosa si está activa. De esta forma, se monitorean todos los paquetes de datos que pasan a través de la red.

Ejecuta "ip link show | grep promisc", que busca interfaces con el flag PROMISC activado. Devuelve "OK" si hay interfaces en modo promiscuo o "Not Found" si no hay ninguna. Estas son las demás funciones presentes en este módulo:

```
    ipv6_status = check_ipv6_configuration()
    dns_status, dns_servers = check_dns_servers()
    dnssec_status = check_dnssec_support()
    gateway_status = check_default_gateway()
    listening_ports = get_listening_ports()
    promiscuous_interfaces = check_promiscuous_interfaces()
    waiting_connections = check_waiting_connections()
    dhcp_status = check_dhcp_client_status()
    arp_monitor_status = check_arp_monitoring_software()
    uncommon_protocols = check_uncommon_network_protocols()
```

Figura 18: Código network.py 2

- ***advanced_network_security.py***

```
    def check_iptables_rules():
    """Verifica las reglas de iptables y si permiten tráfico no deseado."""
    try:
          iptables_rules = subprocess.getoutput("sudo iptables -L -n")
```

Figura 19: Código advanced_network_security.py 1

Verifica las reglas de iptables y si permiten tráfico no deseado, si no está bien configurado, el sistema puede estar expuesto incluso con pocos servicios activos. Estas reglas, nos permiten controlar el tráfico de red de un servidor, permitiendo filtrar los paquetes de red. El usuario debe indicar al firewall el tipo de paquetes entrantes, salientes, los puertos permitidos, el protocolo, y otros datos relacionados con las conexiones de red.

Ejecuta el comando "sudo iptables -L -n" para obtener la lista de reglas activas. Si encuentra reglas con ACCEPT, indica que puede haber tráfico permitido que requiere revisión, si no hay reglas sospechosas, devuelve "No suspicious rules found".

```
      def check_ufw_status():
      """Verifica el estado de UFW (Uncomplicated Firewall)."""
      try:
            ufw_status = subprocess.getoutput("sudo ufw status verbose")
```

Figura 20: Código advanced_network_security.py 2

Verifica el estado del firewall UFW (Uncomplicated Firewall) [49], frente a otras comprobaciones más técnicas, esta función proporciona una visión de alto nivel del estado actual del cortafuegos. El objetivo al instalar este firewall es para poder declarar las reglas iptables y proteger las conexiones. Ejecuta "sudo ufw status verbose" para obtener información detallada sobre el estado del firewall, si UFW está activo, devuelve "UFW: [ACTIVE] with rules applied" y si UFW está inactivo o sin reglas configuradas, devuelve "UFW: [INACTIVE] or no rules".

```
      def check_vpn_usage():
      """Verifica si se está utilizando una VPN o mecanismos de seguridad
de red."""
      try:
            vpn_status = subprocess.getoutput("ip a | grep tun")
```

Figura 21: Código advanced_network_security.py 3

Verifica si hay una VPN (Virtual Private Network) [50] activa o mecanismos de seguridad de red en uso. Una conexión VPN nos permite crear una red local a través de internet,

sin que los integrantes estén conectados entre sí. La conexión estará cifrada mediante un túnel de datos, por lo que la dirección IP del dispositivo será la del servidor VPN.

Ejecuta "ip a | grep tun" para buscar interfaces de red asociadas a túneles VPN.Si encuentra interfaces activas (tun), indica que una VPN está en uso y si no detecta interfaces VPN activas, devuelve "No active VPN interfaces found".

Estas son otras de las funcionalidades presentes en este módulo.

```
    iptables_result = check_iptables_rules()
    ufw_status = check_ufw_status()
    vpn_status = check_vpn_usage()
    ipsec_status = check_ipsec_usage()
```

Figura 22: Código advanced_network_security.py 4

4.1.2.4 Seguridad del sistema y accesos

- ***users_groups_auth.py***

```
    def check_admin_accounts():
    """Verifica las cuentas de administrador."""
    try:
        admins = subprocess.getoutput("getent group sudo")
```

Figura 23: Código users_groups_auth.py 1

Comprueba si existen cuentas en el grupo sudo, lo que indica usuarios con privilegios elevados, estos usuarios pueden ejecutar cualquier comando y modificar cualquier archivo. Por lo que la monitorización de estos usuarios es esencial en cualquier tipo de auditoría, siendo unas de las partes más importantes de la misma.

Para ello usa el comando "*getent group sudo*", que devuelve "Found" si hay cuentas en el grupo sudo y "Not Found" si no hay cuentas con privilegios administrativos.

```
    def check_password_file_consistency():
    """Verifica la consistencia del archivo de contraseñas."""
    try:
        passwd_check = subprocess.getoutput("pwck -r")
```

Figura 24: Código users_groups_auth.py 2

Verifica la integridad del archivo /etc/passwd mediante es uso del comando "pwck -r". Este archivo es uno de los más importantes del sistema ya que si un atacante consigue acceder a él podría comprometer la seguridad de las contraseñas y hacerse con privilegios de sudo. [51] Devuelve "OK" si es consistente."Not Consistent" si hay problemas.

```
    def check_password_hashing_methods():
    """Verifica los métodos de hashing de contraseñas."""
    try:
        shadow_file = subprocess.getoutput("cat /etc/shadow")
```

Figura 25: Código users_groups_auth.py 3

Revisa si las contraseñas están protegidas con SHA-512 o SHA-256. Ambos son algoritmos de hashing de contraseñas, que se utilizan para cifrarlas y hacerlas más seguras. Su funcionamiento consiste en convertir la contraseña en un texto cifrado, y dependiendo del algoritmo utilizado se realiza de una forma u otra. Los algoritmos de la familia SHA (Secure Hash Algorithm), se conocen por ser de los más seguros.

La mayor diferencia entre SHA-256 y SHA-512 es que la salida del cifrado del primero es de 256 bits y del segundo 512, lo que ofrece mayor seguridad pero a su vez un número más elevado de recursos informáticos. [52]

```
    admin_accounts = check_admin_accounts()
    unique_uids = check_unique_uids()
    group_file_consistency = check_group_file_consistency()
    password_file_consistency = check_password_file_consistency()
    password_hashing_methods = check_password_hashing_methods()
    password_hashing_rounds = check_password_hashing_rounds()
    system_users = check_system_users()
    nis_auth_support = check_nis_authentication_support()
    pam_configuration_files = check_pam_configuration_files()
    locked_accounts = check_locked_accounts()
    expired_passwords = check_expired_passwords()
    user_password_aging = check_user_password_aging()
    single_user_mode_auth = check_single_user_mode_authentication()
```

Figura 26: Código users_groups_auth.py 4

- ***file_permissions.py***

```
      def check_file_permissions(file_path):
      """Verifica los permisos de un archivo y devuelve su estado."""
      try:
            if os.path.exists(file_path):
                  file_stat = os.stat(file_path)
                  permissions = stat.filemode(file_stat.st_mode)
            if   file_path   in   ["/boot/grub/grub.cfg",   "/etc/crontab",
"/etc/group", "/etc/passwd"]:
```

Figura 27: Código file_permissions.py 1

Verifica los permisos de un archivo y devuelve su estado. Comprueba si el archivo existe *(os.path.exists(file_path))*, obtiene los permisos del archivo con *os.stat(file_path)*, convierte los permisos en un formato legible *(stat.filemode(file_stat.st_mode))*, si el archivo es crítico (grub.cfg, crontab, group, passwd), lo marca como [OK], de lo contrario, sugiere revisión [SUGGESTION] y si el archivo no existe, devuelve "not found".

```
      def check_directory_permissions(directory_path):
      """Verifica los permisos de un directorio y devuelve su estado."""
      try:
            if os.path.isdir(directory_path):
                  dir_stat = os.stat(directory_path)
                  permissions = stat.filemode(dir_stat.st_mode)
                  if directory_path in ["/root/.ssh", "/etc/cron.d"]:
```

Figura 28: Código file_permissions.py 2

Verifica los permisos de un directorio y devuelve su estado. Comprueba si el directorio existe *(os.path.isdir(directory_path))*, obtiene los permisos con *os.stat(directory_path)*, convierte los permisos a un formato legible *(stat.filemode(dir_stat.st_mode))*, si el directorio es crítico (/root/.ssh, /etc/cron.d), lo marca como [OK], de lo contrario, sugiere revisión [SUGGESTION] y si el directorio no existe, devuelve "not found".

- ***sudo.py***

```
    def check_sudoers_permissions():
    """Verifica los permisos del archivo sudoers y relacionados."""
    try:
        sudoers_permissions = subprocess.getoutput("ls -l
/etc/sudoers")
        sudoers_d_permissions = subprocess.getoutput("ls -l
/etc/sudoers.d")
        readme_permissions = subprocess.getoutput("ls -l
/etc/sudoers.d/README")
```

Figura 29: Código sudo.py 1

Esta función verifica los permisos de los archivos relacionados con la configuración de sudoers. Si los archivos sudoers tienen permisos incorrectos, un atacante podría modificarlos y obtener acceso administrativo al sistema, por lo que es muy importante revisar correctamente los permisos de estos archivos.

Los archivos que se revisan son:

- /etc/sudoers → Archivo principal de configuración de sudo.
- /etc/sudoers.d → Directorio que puede contener configuraciones adicionales de sudo.
- /etc/sudoers.d/README → Archivo de documentación sobre la configuración de sudo.

Ejecuta "ls -l /etc/sudoers", "ls -l /etc/sudoers.d", y "ls -l /etc/sudoers".d/README, que listan los permisos de estos archivos, comprueba si el usuario propietario es root y si los permisos son adecuados. Finalmente, si todo está correcto, devuelve "OK", de lo contrario, "Warning".

```
    def check_sudoers_with_guid_bit():
    """Checks for files with the setgid bit in the sudoers directory or
related files."""
    try:
        sudoers_with_guid = subprocess.getoutput("find /etc/sudoers* -
type f -exec ls -l {} + 2>/dev/null | grep 's' ")
```

Figura 30: Código sudo.py 2

Busca archivos relacionados con sudoers que tengan el bit setgid [53] activado, esta configuración permite que cualquier usuario obtenga privilegios de root, rompiendo el modelo de privilegios mínimos, tanto esta función como módulo al completo son uno de los pilares básicos de nuestra herramienta, usando también la información obtenida para la realización del hackatón.

Usa "find /etc/sudoers* -type f -exec ls -l {} + 2>/dev/null | grep 's' " para encontrar archivos con el bit setgid activado, si encuentra archivos con setgid, advierte sobre posibles problemas de seguridad y si no encuentra archivos con setgid, indica que el sistema es seguro en este aspecto.

```
      def check_files_with_suid_or_guid():
      """Busca archivos con bits setuid (SUID) o setgid (GUID) en todo el
sistema."""
      try:
            result = subprocess.getoutput(
                  "find / -perm /6000 -type f -exec ls -l {} + 2>/dev/null"
            )
```

Figura 31: Código sudo.py 3

Busca archivos en todo el sistema con los bits setuid o setgid, que pueden representar riesgos de seguridad. Set User ID (setuid) [54], como explicamos anteriormente, es un bit que permite a un usuario normal ejecutar un archivo o un programa con privilegios de usuario root. Es útil para que el usuario adquiera permisos de root por para un tiempo determinado, pero puede ser peligroso ya que un usuario puede elevar privilegios de forma maliciosa. Esta función en concreto formará parte del hackatón, siendo uno de los focos de ataque por parte de los participantes.

Usa "find / -perm /6000 -type f -exec ls -l {} + 2>/dev/null" para buscar archivos con permisos setuid o setgid. Si encuentra archivos con estos permisos, muestra una advertencia de seguridad con la lista de archivos encontrados, si no encuentra archivos con setuid o setgid, indica que el sistema es seguro en este aspecto.

```
      sudo_access = check_sudo_access()
```

```
    sudoers_file = check_sudoers_file()
    sudoers_inclusions = check_sudoers_inclusions()
    sudoers_with_guid = check_sudoers_with_guid_bit()
    system_files_with_suid_guid = check_files_with_suid_or_guid()
```

Figura 32: Código sudo.py 4

- ***service_accounts.py***

```
    def check_service_accounts_with_elevated_privileges():
        """Verifica si hay cuentas de servicio con privilegios elevados
(sudo/root)."""
        try:
            sudo_accounts = subprocess.getoutput("grep -E 'sudo|root'
/etc/group")
```

Figura 33: Código service_accounts.py 1

Verifica si hay cuentas de servicio con privilegios elevados (sudo o root), las cuentas de servicio deben de tener privilegios mínimos, por lo que si estas cuentas tienen este tipo de privilegios sería un problema grande de seguridad, debido a esto verificar si existen es esencial.

Usa "grep -E 'sudo|root' /etc/group" para buscar usuarios en los grupos sudo y root. Si encuentra cuentas con acceso privilegiado, devuelve un mensaje indicando que hay cuentas con privilegios elevados, si no encuentra ninguna, devuelve un mensaje indicando que no hay cuentas de servicio con privilegios elevados.

```
    def check_encrypted_passwords_for_service_accounts():
    """Verifica que las contraseñas de las cuentas de servicio estén
cifradas correctamente."""
        try:
            shadow_file = subprocess.getoutput("cat /etc/shadow")
            if shadow_file:
                accounts = shadow_file.splitlines()
                unencrypted_accounts = []
                for account in accounts:
                    username = account.split(":")[0]
```

```
                    password = account.split(":")[1]
                    if password == "":  # Contraseña vacía (sin cifrar)
                        unencrypted_accounts.append(username)
```

Figura 34: Código service_accounts.py 2

Verifica que las contraseñas de las cuentas de servicio estén cifradas y no almacenadas en texto claro, si una cuenta tiene una contraseña sin cifrar o vacía podría autenticarse sin ningún tipo de control, por ello /etc/shadow debe de estar perfectamente controlado.

Usa "cat /etc/shadow" para leer el archivo /etc/shadow, que almacena las contraseñas cifradas, divide el archivo en líneas y revisa cada cuenta y extrae el nombre de usuario y la contraseña de cada línea. Si la contraseña está vacía (""), significa que no está cifrada y la cuenta se marca como insegura, si encuentra cuentas con contraseñas no cifradas, devuelve la lista de nombres de usuario afectados y si todas las cuentas tienen contraseñas cifradas, devuelve un mensaje indicando que el sistema es seguro en este aspecto.

```
elevated_privileges = check_service_accounts_with_elevated_privileges()
encrypted_passwords = check_encrypted_passwords_for_service_accounts()
inactive_accounts = check_inactive_service_accounts()
```

Figura 35: Código service_accounts.py 3

- ***security_policies.py***

```
    def check_password_policy():

    """Verifica las políticas de contraseñas configuradas en el
sistema."""
        try:
            login_defs = subprocess.getoutput("cat /etc/login.defs")
            password_policy = {}

            # Longitud mínima de la contraseña
            min_length = subprocess.getoutput("grep -i 'PASS_MIN_LEN'
/etc/login.defs")
```

```
            password_policy["Min Length"] = min_length.split()[-1] if
min_length and not min_length.startswith("#") else "Not Set or Commented"

            # Edad mínima de la contraseña
            min_age = subprocess.getoutput("grep -i 'PASS_MIN_DAYS'
/etc/login.defs")
            password_policy["Min Age"] = min_age.split()[-1] if min_age
and not min_age.startswith("#") else "Not Set or Commented"

            # Edad máxima de la contraseña
            max_age = subprocess.getoutput("grep -i 'PASS_MAX_DAYS'
/etc/login.defs")
            password_policy["Max Age"] = max_age.split()[-1] if max_age
and not max_age.startswith("#") else "Not Set or Commented"

            # Advertencia antes de expiración de la contraseña
            warn_age = subprocess.getoutput("grep -i 'PASS_WARN_AGE'
/etc/login.defs")
            password_policy["Warn Age"] = warn_age.split()[-1] if
warn_age and not warn_age.startswith("#") else "Not Set or Commented"

            return password_policy
        except Exception as e:
            return f"Error al verificar políticas de contraseñas:
{str(e)}"
```

Figura 36: Código security_policies.py 1

Verifica las políticas de contraseñas configuradas en el sistema, imponiendo unos requisitos mínimos adecuados, cumplimentando marcos como el ISO 27001 que requiere una longitud mínima y frecuencia de cambio.

Lee el archivo /etc/login.defs, que contiene configuraciones de seguridad relacionadas con contraseñas, extrae y analiza las siguientes configuraciones:

- Longitud mínima de la contraseña (PASS_MIN_LEN)
- Edad mínima de la contraseña (PASS_MIN_DAYS)

- Edad máxima de la contraseña (PASS_MAX_DAYS)
- Días de advertencia antes de que la contraseña expire (PASS_WARN_AGE)

```
    def check_audit_logs():
        """Verifica que los logs de auditoría estén habilitados para
acciones de usuario y sistema."""
        try:
            #     Verificar     configuración     de     auditoría     en
/etc/audit/auditd.conf
            audit_config           =           subprocess.getoutput("cat
/etc/audit/auditd.conf")

            # Verificar que la auditoría esté activada
            audit_status = "Auditd configured correctly" if "active = yes"
in audit_config else "Auditd not properly configured"

            # Verificar que la auditoría se registre en un archivo
            log_file_check  =  subprocess.getoutput("grep  -i  'log_file'
/etc/audit/auditd.conf")
            audit_status  +=  ",  Log  file  configured  correctly"  if
"log_file" in log_file_check else ", Log file not configured properly"
            return audit_status
        except Exception as e:
       return f"Error al verificar configuración de logs de auditoría:
{str(e)}"
```

Figura 37: Código security_policies.py 2

Verifica si los logs de auditoría están correctamente configurados en el sistema, si no están activos no hay forma de registrar incidentes, representando un grave problema para la detección de amenazas y consecuentemente no funcionarían las herramientas monitoring_tool ni defense_tool, ya que utilizan este tipo de logs para su uso.

Lee el archivo /etc/audit/auditd.conf, que controla la configuración del servicio auditd, comprueba si la auditoría está activada (active = yes), verifica si se ha configurado un

archivo para registrar los logs (log_file) y por último devuelve un mensaje indicando si la auditoría y el archivo de logs están configurados correctamente.

```
    password_policy = check_password_policy()
    audit_logs = check_audit_logs()
```

Figura 38: Código security_policies.py 3

- ***app_security.py***

```
    def check_apache_security():
        """Verifica la configuración de seguridad de Apache."""
        try:

            ssl_module = subprocess.getoutput("apachectl -M | grep ssl")
            if ssl_module:
                ssl_status = "SSL module enabled"
            else:
                ssl_status = "SSL module not enabled"

            ssl_port = subprocess.getoutput("ss -tuln | grep :443")
```

Figura 39: Código app_security.py 1

Verifica la configuración de seguridad de Apache [55], el cual es un servidor web de código abierto que permite utilizar internet a través suyo. Esta comprobación es clave porque verifica si el servidor está cifrando las comunicaciones, impidiendo la intercepción de datos.

Comprueba si el módulo SSL está habilitado en Apache "apachectl -M | grep ssl", si el módulo SSL está presente, indica que SSL está habilitado. Posteriormente verifica si Apache está escuchando en el puerto 443 (HTTPS) "ss -tuln | grep :443" y finalmente si el puerto 443 está en uso, confirma que Apache está configurado con HTTPS.

```
    def check_nginx_security():
        """Verifica la configuración de seguridad de Nginx."""
        try:
```

```
            ssl_config       =       subprocess.getoutput("grep       ssl
/etc/nginx/nginx.conf")
```

Figura 40: Código app_security.py 2

Verifica la configuración de seguridad de Nginx [56], este es un servidor web de código abierto que se utiliza como un proxy inverso y balanceador de carga, entre otros. Ayuda a los servidores HTTP a mantenerse estables y alcanzar su mayor eficiencia.

```
    def check_mysql_security():
        """Verifica la configuración de seguridad de MySQL."""
        try:
            # Verifica si MySQL tiene habilitada la opción de conexiones
seguras
            secure_connections  =  subprocess.getoutput("mysql  -e  'SHOW
VARIABLES LIKE \"have_openssl\";'")
            if "YES" in secure_connections:
                ssl_status = "SSL enabled"
            else:
                ssl_status = "SSL not enabled"

            # Verifica si MySQL tiene configurado un usuario 'root' con
acceso remoto
            root_access  =  subprocess.getoutput("mysql  -e  'SELECT  host,
user FROM mysql.user WHERE user=\"root\";'")
            if "localhost" in root_access:
                root_access_status = "Root access limited to localhost"
            else:
                root_access_status  =  "Root  access  is  not  limited  to
localhost"
    return    f"MySQL    SSL    Status:    {ssl_status},    Root    access:
{root_access_status}"
```

Figura 41: Código app_security.py 3

Verifica la configuración de seguridad de MySQL [57], que es un software de código abierto que se usa para guardar datos de diferentes bases de datos relacionales. SQL (Structured Query Language) es el lenguaje utilizado en MySQL para realizar las

diferentes operaciones de las bases de datos, como insertar, eliminar, etc. La implementación de esta configuración de seguridad asegura que el usuario root solo tenga acceso desde localhost, evitando la extracción de información presente en la base de datos.

Comprueba si MySQL tiene SSL habilitado "mysql -e 'SHOW VARIABLES LIKE \"have_openssl\";"'. Si devuelve "YES", significa que SSL está activado en MySQL. A continuación verifica si el usuario root tiene acceso remoto "mysql -e 'SELECT host, user FROM mysql.user WHERE user=\"root\";"' Si solo permite acceso desde localhost, la configuración es segura.

```
    def check_ssl_configurations():
        """Verifica la configuración de SSL/TLS en los servidores web y
servicios."""
        try:
            apache_ssl = check_apache_security()
            nginx_ssl = check_nginx_security()
            ssl_certificates = subprocess.getoutput("ls /etc/ssl/certs")
            return f"{apache_ssl}, {nginx_ssl}, {ssl_status}"
```

Figura 42: Código app_security.py 4

Verifica configuraciones generales de SSL/TLS, el cual es un sistema que se encarga de verificar la identidad del usuario y cifrar la conexión de red con el protocolo Secure Sockets Layer/Transpor Layer Security (SSL/TLS). Es importante ya que protege los datos privados y refuerza la confianza entre los clientes y navegadores, entre otros.

Llama a las funciones check_apache_security() y check_nginx_security() para verificar la configuración de SSL en servidores web. Verifica si existen certificados SSL en el sistema y si hay certificados en /etc/ssl/certs, indica que existen estos certificados SSL configurados.

```
    apache_security = check_apache_security()
    nginx_security = check_nginx_security()
    mysql_security = check_mysql_security()
    postgresql_security = check_postgresql_security()
    ssl_configurations = check_ssl_configurations()
```

Figura 43: Código app_security.py 5

4.1.2.5 Archivos y directorios

- ***home_directories.py***

```
    def check_home_directory_permissions():
        """Verifica los permisos de los directorios de inicio de los
usuarios."""
        try:
            home_dirs = [f"/home/{user}" for user in os.listdir('/home')
if os.path.isdir(f"/home/{user}")]
            permissions = {}

            for home in home_dirs:
                home_stat = os.stat(home)
                perm = stat.filemode(home_stat.st_mode)
                if perm == "-rwx------":
```

Figura 44: Código home_directories.py 1

Verifica los permisos de los directorios de inicio, revisando la ruta /home/usuario de cada usuario del sistema, la importancia de esta funcionalidad reside en que si este directorio tiene permisos amplios, otros usuarios del sistema podrían listar su contenido, exponiendo así datos sensibles y rompiendo el aislamiento de usuarios.

Lista los directorios en /home/ que sean válidos (os.listdir('/home') y os.path.isdir())., obtiene los permisos con os.stat(home).st_mode y los convierte en un formato legible con stat.filemode(). Si los permisos son -rwx------ (700), se marcan como [OK]; de lo contrario, [SUGGESTION].

```
    def check_shell_history_files():
        """Verifica la existencia y permisos de los archivos de historial
de shell."""
        try:
            home_dirs = [f"/home/{user}" for user in os.listdir('/home')
if os.path.isdir(f"/home/{user}")]
            history_files = {}
```

```
            for home in home_dirs:
                bash_history = os.path.join(home, ".bash_history")
                zsh_history = os.path.join(home, ".zsh_history")
                history_permissions = "OK"
                # Comprobar si los archivos existen y sus permisos
                if os.path.exists(bash_history):
                    bash_stat = os.stat(bash_history)
                    bash_perm = stat.filemode(bash_stat.st_mode)
                    if bash_perm != "-rw-------":    # Permisos 600
recomendados
                        history_permissions = "SUGGESTION"
                    history_files[bash_history] = f"{bash_perm} [OK]" if
history_permissions == "OK" else f"{bash_perm} [SUGGESTION]"

                if os.path.exists(zsh_history):
                    zsh_stat = os.stat(zsh_history)
                    zsh_perm = stat.filemode(zsh_stat.st_mode)
                    if zsh_perm != "-rw-------":    # Permisos 600
recomendados
                        history_permissions = "SUGGESTION"
                    history_files[zsh_history] = f"{zsh_perm} [OK]" if
history_permissions == "OK" else f"{zsh_perm} [SUGGESTION]"

            return history_files
        except Exception as e:
            return f"Error al verificar archivos de historial de shell:
{str(e)}"
```

Figura 45: Código home_directories.py 2

Verifica la existencia y permisos de los archivos .bash_history y .zsh_history [58]. En estos archivos ocultos, se guardan los comandos que se han ido ejecutando en la terminal. Algunas veces en estos comando almacenan contraseñas, tokens y rutas importantes; este historial es parte de la trazabilidad del usuario por lo que su protección es primordial para evitar que otros lo lean o manipulen los comandos ejecutados.

Busca los archivos .bash_history y .zsh_history en los directorios de /home/, si existen, obtiene los permisos con os.stat() y los convierte con stat.filemode(). Si los permisos no son -rw------- (600), recomienda cambiarlo.

```
    home_permissions = check_home_directory_permissions()
    home_ownership = check_home_directory_ownership()
    history_files = check_shell_history_files()
```

Figura 46: Código home_directories.py 3

- ***storage_device.py***

```
    def check_disk_encryption():
        """Verifica si el cifrado de discos (LUKS) está habilitado en los
discos críticos."""
        try:
            # Verificar si los discos están cifrados con LUKS
            encrypted_disks = subprocess.getoutput("lsblk -f | grep luks")
```

Figura 47: Código storage_device.py 1

Verifica si el cifrado de discos (LUKS) [59] está habilitado, este cifrado es el estándar de cifrado de disco completo en Linux. Se encarga de proteger los discos mediante un cifrado, obligando al usuario a autenticarse de una forma determinada, si esto no fuese así cualquier atacante con acceso físico al disco podría extraer todo su contenido fácilmente. Esta funcionalidad cubre tanto la seguridad física y lógica con un comando que aporta un gran valor.

Utiliza "lsblk -f | grep luks" para comprobar si existen particiones cifradas. Si encuentra discos cifrados, devuelve "Disk encryption enabled: ...", si no hay discos cifrados, devuelve "No disk encryption enabled.".

```
    def check_filesystem_integrity():
        """Verifica la integridad del sistema de archivos (usando
fsck)."""
        try:
            fs_check = subprocess.getoutput("sudo fsck -n /dev/sda1")
```

Figura 48: Código storage_device.py 2

Mientras la mayoría de las funciones examinan la configuración, esta verifica la integridad real del sistema de archivos con fsck, siendo una de las verificaciones más importantes ya que se basa en la integridad y no en archivos de configuración o políticas.

Usa "sudo fsck -n /dev/sda1" para realizar una verificación de solo lectura en el sistema de archivos. Si el sistema de archivos está limpio, devuelve "Filesystem integrity is OK.", si encuentra errores, devuelve "Filesystem errors found or need checking.".

```
    disk_encryption = check_disk_encryption()
    mount_options = check_mount_options()
    disk_health = check_disk_health()
    fs_integrity = check_filesystem_integrity()
```

Figura 49: Código storage_device.py 3

4.1.2.6 Procesos, memoria y actividad

- ***mem_process.py***

```
    def check_meminfo():
        """Verifica el archivo /proc/meminfo."""
        try:
            meminfo = subprocess.getoutput("cat /proc/meminfo")
            return "Found" if meminfo else "Not Found"
        except Exception as e:
            return f"Error al verificar /proc/meminfo: {str(e)}"
```

Figura 50: Código mem_process.py 1

Verifica si el archivo /proc/meminfo existe y tiene datos. Este archivo nos permite ver cómo se están utilizando los diferentes recursos de memoria y cuáles son.

Ejecuta cat /proc/meminfo, que muestra información sobre la RAM y el intercambio (swap). Si el comando tiene salida (meminfo no está vacío), devuelve "Found", si no hay salida, devuelve "Not Found".

```
    def check_dead_zombie_processes():
        """Busca procesos muertos/zombie."""
```

```
        try:
            zombie_processes = subprocess.getoutput("ps aux | grep 'Z'")
            return "Found" if zombie_processes else "Not Found"
        except Exception as e:
            return f"Error al buscar procesos zombie: {str(e)}"
```

Figura 51: Código mem_process.py 2

Su objetivo es detectar procesos zombie (procesos terminados cuyos padres no los han eliminado correctamente). Es muy importante debido a que indicaría una mala gestión de procesos ocupando recursos del sistema y si se excedieran con el número de procesos se podría agotar la tabla de procesos pudiendo colapsar el lanzamiento de nuevos procesos legítimos.

Ejecuta "ps aux | grep 'Z'", que busca procesos con estado "Z" (zombie). Si encuentra procesos zombie, devuelve "Found", si no hay, devuelve "Not Found".

```
    meminfo = check_meminfo()
    zombie_processes = check_dead_zombie_processes()
    io_waiting_processes = check_io_waiting_processes()
    prelink_tooling = check_prelink_tooling()
```

Figura 52: Código mem_process.py 3

- ***logs.py***

```
    def check_old_logs(log_file, days_threshold=30):
        """Verifica si los logs son antiguos (más de 30 días por
defecto)."""
        try:
            if os.path.exists(log_file):
                file_mod_time = os.path.getmtime(log_file)
                days_old = (time.time() - file_mod_time) / (60 * 60 * 24)
                if days_old > days_threshold:
                    return f"{log_file}: [OLD] ({days_old:.2f} days)"
                else:
                    return f"{log_file}: [OK] ({days_old:.2f} days)"
            else:
```

```
                return f"{log_file}: [NOT FOUND]"
        except Exception as e:
            return f"Error al verificar la antigüedad de {log_file}:
{str(e)}"
```

Figura 53: Código logs.py 1

Esta función determina si un archivo de log [60] tiene más de 30 días (o un umbral definido). Si la valoración es negativa podría ser por varios motivos a revisar, causando un abandono de logs y llenando el espacio del disco, incumpliendo así las políticas establecidas para la retención de los mismos.

Usa "os.path.getmtime(log_file)" para obtener la última fecha de modificación del archivo, después cuántos días han pasado desde la última modificación. Si el archivo es más antiguo que el umbral, devuelve [OLD], si aún está dentro del rango, devuelve [OK].

```
    log_files = ["/var/log/auth.log", "/var/log/syslog",
"/var/log/messages", "/var/log/dmesg"]
        # Verificación de existencia y estado de los archivos de log
        for log_file in log_files:
            print(check_log_file_exists(log_file))
            print(check_log_integrity(log_file))
            print(check_old_logs(log_file))
        # Verificación de la configuración de rotación de logs
        print(check_log_rotation())
```

Figura 54: Código logs.py 2

4.1.2.7 Contenedores y entorno virtual

- ***containers_security.py***

```
    def check_docker_container_configuration():
        """Verifica la configuración de los contenedores Docker."""
        try:
            # Comprobar contenedores en ejecución
            containers_running = subprocess.getoutput("docker ps")
            if "CONTAINER ID" in containers_running:
```

```
                containers = containers_running.splitlines()[1:]  # Ignorar encabezado
                container_info = []
                for container in containers:
                    container_id = container.split()[0]
                    container_ports = subprocess.getoutput(f"docker port {container_id}")
                    container_resources = subprocess.getoutput(f"docker inspect {container_id} --format '{{{{.HostConfig.Memory}}}}'")

                    container_info.append({
                        "container_id": container_id,
                        "ports": container_ports,
                        "memory_limit": container_resources
                    })

                return container_info
            else:
                return "No Docker containers running."
        except Exception as e:
        return f"Error al verificar configuración de contenedores Docker: {str(e)}"
```

Figura 55: Código containers_security.py 1

Verifica la configuración de los contenedores Docker en ejecución, permitiendo ver que contenedores están funcionando, que puertos tienen abiertos y su límite de recursos. Si un contenedor excede los límites de memoria o expone puertos sensibles puede ser un vector de ataque usual, por lo que verificar esta configuración es necesario. Usa docker ps para listar los contenedores en ejecución. Si hay contenedores activos, obtiene:

- container_id (ID del contenedor).
- docker port {container_id} para ver los puertos asignados.
- docker inspect {container_id} --format '{{.HostConfig.Memory}}' para conocer los límites de memoria.

```
    def check_container_vulnerabilities():
        """Verifica si las imágenes de contenedores tienen
vulnerabilidades conocidas."""
        try:
            vulnerability_scan = subprocess.getoutput("docker scan --
all")
```

Figura 56: Código containers_security.py 2

Verifica si las imágenes de los contenedores tienen vulnerabilidades conocidas (CVEs), estos contenedores pueden tener configuraciones antiguas de todo tipo, exponiéndose así a numerosos fallos de seguridad. El valor añadido de esta función se muestra en que va más allá de la configuración y analiza vulnerabilidades reales ya existentes en el sistema.

Usa "docker scan --all" para escanear todas las imágenes en busca de vulnerabilidades. Si el escaneo detecta vulnerabilidades, devuelve "Vulnerabilities found in Docker images.", si no se encuentran vulnerabilidades, devuelve "No vulnerabilities found in Docker images.".

```
    docker_configuration = check_docker_container_configuration()
    lxc_configuration = check_lxc_container_configuration()
    images_sources = check_container_images_sources()


    vulnerabilities = check_container_vulnerabilities()
```

Figura 57: Código containers_security.py 3

4.1.2.8 Backup y recuperación

- **backup.py**

```
    def check_backup_storage():
        """Verifica si las copias de seguridad están almacenadas de forma
segura."""
        try:
            backup_dir = "/backup"
            if os.path.exists(backup_dir) and os.access(backup_dir,
os.R_OK):
```

```
                return f"Backup directory found: {backup_dir}\nEnsure the
backup directory is secure and properly configured."
            else:
                return f"Backup directory not found or not accessible at
{backup_dir}.\nEnsure that the backup directory exists and is properly
secured."
        except Exception as e:
            return f"Error checking backup storage: {str(e)}"
```

Figura 58: Código backup.py 1

Verifica si las copias de seguridad están almacenadas en un directorio seguro, además que si tenemos una política o herramienta de backup y no hay ruta física donde almacenarlo sería incongruente.

Comprueba si el directorio /backup existe y es accesible. Si el directorio existe, informa que está disponible, si no existe o no es accesible, advierte que se debe configurar adecuadamente.

```
    def check_restore_process():
        """Verifica si el proceso de restauración de las copias de
seguridad está configurado y es válido."""
        try:
            restore_script = "/usr/local/bin/restore_backup.sh"  # Ruta
de un script de restauración
```

Figura 59: Código backup.py 2

Verifica si existe un proceso de restauración de copias de seguridad y si es ejecutable, debido a que hacer el backup es igual de importante que poder restaurarlo, ya que muchas veces existe ese backup pero no hay un procedimiento probado de restauración. Esta funcionalidad valida su efectividad.

Comprueba si el archivo /usr/local/bin/restore_backup.sh existe y tiene permisos de ejecución. Si el script está disponible, informa que el proceso de restauración está configurado y si no existe, recomienda documentar y probar el proceso de restauración.

```
    backup_schedule = check_backup_schedule()
```

```
    backup_storage = check_backup_storage()
    backup_encryption = check_backup_encryption()
    restore_process = check_restore_process()
```

Figura 60: Código backup.py 3

4.1.2.9 Tests

- **debian_tests.py**

```
    def check_system_binaries():
        """Verifica si los binarios necesarios existen en los directorios
especificados."""
        directories = ['/bin', '/sbin', '/usr/bin', '/usr/sbin',
'/usr/local/bin']
        results = {}
        for directory in directories:
            try:
                result = subprocess.getoutput(f"ls {directory}")
```

Figura 61: Código debian_tests.py 1

Verifica la presencia de binarios esenciales en los directorios del sistema, recorre una lista de directorios donde suelen almacenarse los binarios críticos (/bin, /sbin, etc.), usa ls para verificar si hay archivos en esos directorios. Si hay archivos, devuelve "FOUND", de lo contrario, "NOT FOUND".

```
    def check_debian_tests():
        """Verifica la ejecución de pruebas de Debian y sus resultados."""
        try:
            test_results = subprocess.getoutput("debian-tests")
```

Figura 62: Código debian_tests.py 2

Ejecuta pruebas de Debian y verifica los resultados, esto garantiza que el sistema base es funcional y estable, estas pruebas ayudan a detectar anomalías antes de que afecten en mayor escala. Ejecuta el comando debian-tests. Si la salida contiene "DEB-0001", se asume que la prueba se ejecutó correctamente y si no, indica que la prueba no se ejecutó como se esperaba.

```
    system_binaries = check_system_binaries()
    pam_status = check_pam()
    debian_test_results = check_debian_tests()
    required_packages = check_required_packages()
    filesystem_checks = check_filesystem_checks()
```

Figura 63: Código debian_tests.py 3

4.1.2.10 Protección y detección de amenazas

- **malware_protection.py**

```
    def check_rootkit_scan():
        """Realiza un escaneo de rootkits usando chkrootkit o rkhunter."""
        try:
            chkrootkit_status = subprocess.getoutput("which chkrootkit")
            if chkrootkit_status:
                chkrootkit_scan = subprocess.getoutput("sudo chkrootkit")
                return f"Chkrootkit scan results:\n{chkrootkit_scan}"
            else:
                rkhunter_status = subprocess.getoutput("which rkhunter")
                if rkhunter_status:
                    rkhunter_scan = subprocess.getoutput("sudo rkhunter --check")
                    return f"Rkhunter scan results:\n{rkhunter_scan}"
                else:
                    return "No rootkit scanner found. Consider installing chkrootkit or rkhunter for rootkit detection."
        except Exception as e:
            return f"Error performing rootkit scan: {str(e)}"
```

Figura 64: Código malware_protection.py 1

Escanea en busca de rootkits utilizando chkrootkit o rkhunter, dos herramientas especializadas en esta funcionalidad. Los rootkits son uno de los tipos más peligrosos de malwares, pudiendo ocultarse de los logs y procesos normales del sistema ya que se ejecutan a bajo nivel, estos malwares pueden ocultar conexiones, usuarios y permitir

acceso remoto, por ello estas dos herramientas están diseñadas específicamente para detectarlos mediante firmas y análisis heurístico.

Verifica la instalación de estas herramientas y si están instaladas las ejecuta.

```
    def check_antivirus_status():
        """Verifica la configuración de herramientas antivirus, si están
disponibles."""
        try:
            clamav_status = subprocess.getoutput("which clamscan")
            if clamav_status:
                clamav_version = subprocess.getoutput("clamscan --
version")
                return f"ClamAV is installed. Version:\n{clamav_version}"
            else:
                return "ClamAV not installed. Consider installing it for
malware protection."
            # Alternativamente, verificar Sophos (si está instalado)
            sophos_status = subprocess.getoutput("which savscan")
            if sophos_status:
                sophos_version = subprocess.getoutput("savscan --
version")
                return f"Sophos Antivirus is installed.
Version:\n{sophos_version}"
            else:
                return "Sophos Antivirus not installed. Consider
installing it for malware protection."
        except Exception as e:
            return f"Error checking antivirus status: {str(e)}"
```

Figura 65: Código malware_protection.py 2

Verifica si hay herramientas antivirus instaladas y activas,en primer lugar ClamAv que es de código abierto y ampliamente usada y en segundo lugar, Sophos, que es usada en ambientes corporativos y es más avanzada. Esta función es considerada de las más importantes porque pocas herramientas de auditoría analizadas en la fase de investigación analizaban el estado de protección básica contra malware

```
    rootkit_scan = check_rootkit_scan()
    antivirus_status = check_antivirus_status()
    suspicious_memory_behaviors = check_suspicious_memory_behaviors()
```

Figura 66: Código malware_protection.py 3

4.1.3 Aproximaciones fallidas

Durante la creación de Audit-Tool, surgieron diversas dificultades que derivaron en soluciones progresivas. Cada obstáculo permitió detectar limitaciones y ajustar el diseño para robustecer la herramienta. A continuación, se exponen siete aproximaciones que no funcionaron como se esperaba, junto a las soluciones que permitieron avanzar:

1. **Diseño inicial:** En la primera versión se partió de un solo archivo con toda la lógica en un único archivo Python extenso, pensando que sería más sencillo de ejecutar. Esta aproximación fracasó al crecer el código, superando las más de mil líneas de código, complicando el mantenimiento y la detección de errores. Además, si una función fallaba abortaba la ejecución completa de la auditoría. La solución que seguimos fue reorganizar el código a una arquitectura modular, separando así las responsabilidades en archivos independientes para aislar los posibles errores y facilitando la gestión del código.
2. **Carga estática de módulos:** Al modularizar el código, inicialmente se importaban de forma explícita cada uno de los módulos (llamando explícitamente a import modules.sys_info, import modules.kernel, etc.). Esto funcionaba, pero resultó poco flexible: cada vez que se añadía un módulo era necesario modificar el código principal. Se intentó solucionar con una lista fija de nombres de módulos, pero también era propenso a olvidos. Esta estrategia falló cuando en la versión v1.1 se duplicaron los módulos y nos olvidamos de incorporarlos. La solución adoptada fue implementar una carga dinámica de módulos usando *importlib.import_module*, que recorre automáticamente el directorio modules/ para importar solo los archivos presentes.
3. **Parsing inconsistente de la salida de comandos:** Varias comprobaciones dependían de interpretar la salida de comandos del sistema (por ejemplo, el formato de iptables -L -n o de netstat). En un primer intento, se escribieron

funciones de parsing muy específicas, asumiendo cierto formato fijo de salida. Esto falló al probar la herramienta en diferentes distribuciones Linux donde, netstat no estaba instalado por defecto, o donde los encabezados de columnas variaban por idioma o versión. Como resultado, algunas funciones devolvían falsos positivos o lanzaban excepciones al no encontrar el texto esperado. Se resolvió este problema usando herramientas más estándar como ss, aplicando flags coherentes, y añadiendo expresiones regulares más resistentes a cambios en los encabezados para extraer la información deseada. Además, se añadieron pruebas unitarias con muestras de salida simulada de estos comandos para validar los parsers.

4. **Ejecución parcial ante errores:** En la transición a la arquitectura modular, se descubrió que si un módulo lanzaba una excepción no controlada (por ejemplo, al no encontrar un archivo esperado), la herramienta se detenía abruptamente sin ejecutar los módulos restantes. Esto se consideró un fallo crítico, ya que comprometía la completitud de la auditoría. Inicialmente, se intentó solucionar envolviendo todo el llamado de módulos en un bloque try/except global, pero este enfoque falló porque atrapaba la excepción demasiado tarde (tras abortar la ejecución del módulo problemático sin reportar cuál fue). Finalmente, se modificó el diseño para aislar la ejecución de cada módulo: el programa principal itera por la lista de módulos y ejecuta cada uno en un bloque propio try/except. Si ocurre un error en uno, se registra en el informe un resultado de error para ese módulo (incluyendo el mensaje de excepción) pero la ejecución continúa con el siguiente módulo. De esta forma, la herramienta siempre produce un informe completo con al menos un estado por cada chequeo, incluso si uno o varios módulos fallan a mitad de proceso.

5. **Integración de resultados y formato del informe:** En versiones tempranas, cada módulo imprimía sus hallazgos en pantalla directamente. Se intentó consolidar esto redirigiendo salidas estándar a un archivo, pero el resultado era desordenado y difícil de procesar automáticamente. También se probó a construir manualmente un string con todos los resultados para un informe más legible, pero esto se volvió insostenible al multiplicarse los módulos (escapar

caracteres especiales, gestionar estilos, etc.). La solución fue optar por formato JSON para el informe final: cada módulo devuelve un diccionario Python con sus resultados, y el programa principal agrega estos diccionarios en una estructura unificada que luego vuelca a JSON. Esto facilitó tanto la legibilidad (mediante visores JSON) como la post-procesabilidad (el informe puede ser consumido fácilmente por otras herramientas o scripts).

6. **Problemas de privilegios (sudo):** Durante el desarrollo, a menudo se ejecutaba el código sin privilegios de administrador para probar rápidamente módulos individuales. Muchas funciones fallaban (por ejemplo, consultas a /etc/shadow o llamadas a iptables) debido a permisos denegados. Inicialmente se intentó mitigar capturando la excepción de permiso en cada comando, pero esto se volvió repetitivo y propenso a olvidos. Además, la falta de privilegios podía simplemente dar resultados incompletos sin error explícito (por ejemplo, systemctl list-units sin sudo no lista servicios de sistema). Esto llevó a confusión en el debug. La solución implementada fue añadir al inicio de la ejecución una verificación de permisos de root (p.ej, función check_root() que usa os.geteuid() para comprobar UID 0). Si la herramienta no se lanza con sudo, se aborta inmediatamente mostrando un mensaje claro de error. Esto garantizó que los desarrolladores/usuarios supieran que debían ejecutar con privilegios adecuados, evitando comportamientos inesperados más adelante.

7. **Rendimiento y tiempo de ejecución:** Con la adición de numerosos módulos, se observó que el tiempo total de ejecución de Audit-Tool aumentó notablemente, llegando a ser de varios minutos en sistemas con mucha información (por ejemplo, listados largos de usuarios o servicios). Se intentó primero optimizar reemplazando algunas llamadas a comandos externos por llamadas a librerías Python (por ejemplo, usar psutil para obtener información de procesos en lugar de invocar a ps/top). Sin embargo, esta aproximación fracasó en parte porque la introducción de librerías externas debido a que añadía nuevas dependencias. La solución combinó varias estrategias: se optimizaron algunas consultas (por ejemplo, limitando el alcance de búsqueda de find a rutas críticas en lugar de todo el sistema de archivos), se paralelizó la ejecución de ciertos grupos de

módulos no interdependientes empleando subprocesos Python (threads), mejorando considerablemente el tiempo de ejecución.

4.1.4 Métricas de Complejidad

Para evaluar la complejidad del desarrollo de la herramienta Audit-Tool y cuantificar el esfuerzo implicado, se pueden considerar diversas métricas objetivas. A continuación, se presentan las principales medidas recopiladas, junto con su valor estimado y una justificación de su relevancia:

- **Número de iteraciones/versiones:** Se identifican 4 hitos principales en el desarrollo (prototipo inicial, v1.0 modular, v1.1 ampliada, v2.0 final reorganizada). Cada iteración implicó refactorizar o extender significativamente el código. Este número de versiones es un indicador de complejidad: requirió rediseñar la arquitectura varias veces hasta alcanzar una solución óptima.
- **Número de intentos fallidos:** Además de las versiones formales, se documentaron al menos **7 aproximaciones fallidas** o problemas críticos durante el desarrollo (como los descritos anteriormente). El conteo de estos intentos y correcciones refleja la **complejidad técnica** enfrentada: cada fallo representó tiempo invertido en depurar y reorientar la implementación. Un alto número de intentos indica que la herramienta aborda problemas no triviales que no se resolvieron con la primera solución propuesta.
- **Horas de desarrollo:** Se estima que la construcción de Audit-Tool requirió del orden de 100 a 120 horas de trabajo, distribuidas en análisis, codificación, pruebas y depuración. Esta métrica da cuenta del esfuerzo humano invertido; la alta cifra es consistente con la amplitud de funcionalidades (20 módulos especializados) y las revisiones necesarias tras detectar errores en distintas fases.
- **Líneas de código:** La versión final v2.0 de Audit-Tool contiene aproximadamente 2.400 líneas de código Python (incluyendo comentarios y espacios), repartidas entre el script principal y todos los módulos. Un código de esta extensión conlleva una complejidad en sí mismo, ya que mantener la cohesión y comprensión de tantos componentes requiere una arquitectura clara. El conteo de líneas se emplea como métrica de tamaño del código, relevante porque a mayor

tamaño, típicamente mayor superficie para posibles bugs y mayor esfuerzo de mantenimiento.

- **Commits y cambios registrados:** El repositorio de Audit-Tool acumula 30 commits a lo largo de su desarrollo. Más que un mero recuento, este historial demuestra la granularidad del trabajo: cada confirmación reflejó una mejora o corrección significativa respecto al estado previo. Al limitar los commits a momentos en los que el avance era realmente apreciable, garantizamos un control preciso de la evolución del código y evidenciamos la complejidad inherente al proceso iterativo de refinamiento.
- **Dependencias externas:** Audit-Tool deliberadamente minimizó dependencias de terceros, apoyándose principalmente en utilidades nativas del sistema en lugar de librerías Python adicionales. Solo se usaron librerías estándar (como json, subprocess) y herramientas del sistema. El hecho de no agregar paquetes externos podría interpretarse como menor complejidad en términos de gestión de dependencias; sin embargo, implicó un esfuerzo extra en el desarrollo de funciones propias para tareas que de otro modo podría proporcionar una librería (decisión tomada para reducir riesgos de compatibilidad y mantener la herramienta ligera). Esta métrica resalta la filosofía de diseño: complejidad interna en lugar de complejidad externa de configuración.
- **Cobertura de funcionalidad:** Si bien más cualitativa que cuantitativa, se podría medir la complejidad por la cantidad de comprobaciones de seguridad incluidas. Audit-Tool realiza del orden de 30-40 chequeos distintos del sistema (sumando todos los módulos y subchequeos dentro de ellos). Este número da una idea del amplio espectro cubierto (cuantas más comprobaciones, más casos a considerar y código a mantener). Es relevante porque correlaciona con el objetivo del proyecto: lograr una auditoría exhaustiva implicó lidiar con múltiples aspectos del sistema, aumentando la complejidad global.

En conjunto, estas métricas muestran que el desarrollo de Audit-Tool fue un proceso extenso y no exento de desafíos. La elevada cantidad de líneas de código y horas invertidas, combinada con numerosas iteraciones y ajustes, son indicativos de un proyecto de considerable complejidad. Sin embargo, el resultado final (una herramienta

robusta y modular capaz de auditar comprensivamente un sistema Linux) justifica el esfuerzo, aportando un componente valioso al entorno seguro.

4.2 Monitoring_Tool

Con el objetivo de detectar accesos no autorizados a archivos sensibles durante la celebración del hackatón, se desarrolló una herramienta de monitorización. En el contexto del entorno seguro, nuestra primera idea era que su propósito fuese actuar como un sistema de alerta temprana ante actividades sospechosas que pudieran comprometer la seguridad de las contraseñas o de los archivos sensibles. Finalmente, decidimos que se basase en la detección del acceso a un archivo que actúa como cebo (honeypot) A continuación, se detalla el proceso de desarrollo de esta herramienta, los comandos empleados en su construcción y uso, las dificultades encontradas y las métricas de complejidad asociadas.

4.2.1 Proceso de desarrollo

Al igual que con Audit-Tool, se comenzó creando un repositorio Git separado para la herramienta Monitoring_Tool. Tras git init en un nuevo directorio, se añadieron los archivos iniciales: un script principal (monitor_tarjetas.py) y un archivo README que describía brevemente la herramienta y su uso. El control de versiones siguió buenas prácticas: commits iniciales establecieron la estructura básica del proyecto y los objetivos. Desde el inicio se planificó que esta herramienta funcionaría continuamente en segundo plano, por lo que se incluyó también un archivo de configuración para integrarla como servicio del sistema desde etapas tempranas del desarrollo.

La idea central fue implementar una técnica de honeypot para detectar accesos indebidos a información sensible. En la planificación se decidió crear archivos señuelo que contuvieran datos ficticios, pero apetecibles para un atacante (por ejemplo, una lista de cuentas bancarias o "tarjetas de crédito" falsas almacenadas en texto plano). El nombre del script, monitor_tarjetas.py, alude a uno de esos señuelos (una supuesta lista de números de tarjeta de crédito). El enfoque inicial fue simple: crear el archivo señuelo y luego vigilar periódicamente si era accedido o modificado. En una primera versión, el script se limitaba a comprobar la marca de tiempo de último acceso del archivo señuelo

en intervalos regulares (cada pocos segundos) mediante llamadas a funciones del sistema de ficheros. Esta aproximación permitió un prototipo rápido, pero presentaba limitaciones: dependía de que el sistema actualizase los metadatos de acceso (lo cual puede estar deshabilitado en montajes con noatime por rendimiento), y no proporcionaba detalles sobre quién había accedido al fichero ni cuándo exactamente.

Estas funciones se integraron en el script principal de Monitoring_Tool con un enfoque modular básico: aunque residían en el mismo archivo Python, se definió una función principal run() que invocaba a cada comprobación y recopilaba sus resultados en un diccionario, similar a la filosofía de Audit-Tool pero a menor escala. La salida de monitor_tarjetas.py al ejecutarse manualmente era un informe en formato JSON con el estado de cada chequeo, lo cual facilitó pruebas individuales de la lógica de cada comprobación.

A medida que se probaba la herramienta en entornos simulados, se fueron detectando carencias en la monitorización. Una de las mejoras clave fue migrar desde la simple vigilancia de ficheros por timestamp a un mecanismo más robusto usando las capacidades de auditoría del propio sistema operativo. En versiones posteriores, se aprovechaba **auditd** (el demonio de auditoría de Linux) para registrar eventos de acceso al archivo señuelo. Se configuró una regla de auditoría del kernel (sudo auditctl -w /home/osboxes/Documents/tarjetas_bancarias.txt -p r -k acceso_tarjetas) que registraba cualquier intento de lectura del fichero señuelo con una etiqueta (*key*). De esta forma, en lugar de depender de chequear manualmente el sistema de ficheros, el script pudo consultar los logs de auditd para obtener eventos detallados (incluyendo la identidad del usuario/proceso que accedió al señuelo y la hora exacta). La integración se realizó mediante el comando ausearch: la herramienta ejecuta periódicamente ausearch -k acceso_tarjetas -ts <última marca> para extraer eventos recientes con la clave de auditoría especificada, y analiza la salida. Además de registrar estos eventos, la herramienta también genera una notificación por correo electrónico cuando se detecta un acceso no autorizado. Se implementó un sistema de envío de correos mediante smtplib y SMTP autenticado, que permite al administrador recibir alertas inmediatas con los detalles del incidente (usuario implicado y su ip, archivo accedido, momento del evento, etc.). Para ello, se configuró una cuenta de correo específica con

una contraseña de aplicación, y se documentaron los pasos necesarios para su activación. Esta funcionalidad, que se probó en múltiples escenarios simulados, permitió validar su utilidad en entornos como hackatones, donde la velocidad de respuesta ante una intrusión es clave.

La incorporación de auditd incrementó significativamente la fiabilidad de la monitorización. Para gestionarlo, el desarrollo del script pasó por refactorizaciones: se añadieron funciones para parsear los registros de auditoría y extraer información útil (usuario, UID, IP de origen si disponible, éxito o fracaso del intento de acceso). También fue necesario coordinar la herramienta con la configuración del sistema: durante la instalación, se proveía una regla de auditoría persistente (por ejemplo, añadiendo a /etc/audit/rules.d/ una línea correspondiente al señuelo) y se habilitaba auditd. Esto requirió documentar en el README pasos de instalación adicionales y verificar en entornos de prueba que los eventos de auditoría se generaban correctamente al acceder al señuelo.

La herramienta estaba concebida para ejecutarse continuamente en segundo plano como un *daemon*. Por ello, parte del desarrollo incluyó la creación de un servicio systemd. Se escribió un archivo unit (monitor_tarjetas.service) con la configuración necesaria: ejecución como root, inicio automático al arrancar el sistema, y reinicio en caso de fallo. La herramienta fue adaptada para este modo de uso: por ejemplo, se añadió un pequeño retraso al inicio para esperar que el sistema terminara de arrancar antes de comenzar la monitorización (evitando alertas falsas por accesos durante el boot), y se aseguró de que los mensajes se registraran tanto por pantalla como en un log de archivo para luego poder inspeccionarlos vía journalctl. Durante pruebas, se utilizó systemctl start monitor_tarjetas para lanzar el servicio manualmente y validar su comportamiento, y systemctl enable monitor_tarjetas para configurar su inicio en boot. Esta integración con el sistema consolidó la herramienta como una solución persistente de monitorización.

La versión final consolidó las funcionalidades de monitorización de honeypots en un mismo paquete. Aunque no se dividió en módulos como en Audit-Tool, sí se estructuró el código para claridad. Por ejemplo, se seccionó lógicamente el código en: definiciones de funciones de envío de alerta, definiciones de funciones de monitorización de

ficheros, y la lógica principal de ejecución cíclica. La herramienta en su estado final era capaz de detectar eventos dinámicos sospechosos (accesos a señuelos), alertando de la situación en tiempo real.

Finalmente, hemos incluido un documento como Apéndice F en el que se describe en profundidad la implementación de la herramienta. Este apéndice recoge el diseño de la arquitectura interna y la explicación completa de los códigos y sus funciones.

4.2.2 Estructura y Arquitectura de la herramienta

- **monitor_tarjetas.py**

Este archivo es el programa principal de la herramienta en el que se desarrollan las diferentes funcionalidades definidas en los objetivos previos. Este script monitorea intentos de lectura sobre un archivo específico, simulando el funcionamiento de un honeypot. Si detecta cualquier tipo de acceso registra el evento en un archivo de log, envía un correo de alerta y bloquea el archivo eliminando todos los permisos. Todo esto se ejecuta en bucle cada pocos segundos.

El script comienza importando una serie de bibliotecas esenciales para su funcionamiento, tanto del sistema operativo como de envío de alertas.

```
import subprocess
import os
import time
from datetime import datetime
```

Figura 67: Bibliotecas monitor_tarjetas.py 1

Para control del sistema, tiempo y ejecución de comandos de shell (como ausearch).

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
```

Figura 68: Bibliotecas monitor_tarjetas.py 2

Para el envío de correos, clave para el envío de la alerta en tiempo real.

```
import pwd
```

Figura 69: Bibliotecas monitor_tarjetas.py 3

Traduce UID (User Id) a nombre de usuario. Para la configuración inicial del programa, necesitamos definir la ruta del archivo a monitorear (os.path.expanduser("/home/osboxes/Documents/tarjetas_bancarias.txt"), la clave del correo electrónico utilizada por auditd para identificar los eventos ("acceso_tarjetas") y el intervalo de medida (2 segundos).

Posteriormente hay que registrar los eventos en "logs/accesos.log", incluyendo hora, usuario e IP de intento. También hay que guardar las alertas enviadas para que no se repitan en la variable eventos_detectados. Una vez se han configurado las diferentes variables, hay que implementar las diferentes funciones que componen la lógica de la herramienta.

```
    def registrar_log(usuario, ip):
        mensaje   =   f"[{datetime.now().strftime('%Y-%m-%d   %H:%M:%S')}]
Usuario: {usuario} | IP: {ip}\n"
        try:
            with open(LOG_PATH, "a") as log_file:
                log_file.write(mensaje)
```

Figura 70: Código monitor_tarjetas.py 1

Escribe en el log de texto un registro del acceso, con hora, usuario e IP, y lanza una excepción si no es correcto. Esto es clave para poder saber a qué hora ha sido el acceso y qué usuario lo ha hecho.

```
    def enviar_alerta_gmail(usuario, ip):
        remitente = "pruebasfede1111@gmail.com"
        receptor = "pruebasfede1111@gmail.com"
        asunto = "ALERTA: Acceso al archivo sensible"
        mensaje = f"""
    Se ha detectado un intento de lectura en tarjetas_bancarias.txt


    🔍 Detalles:
    - Usuario: {usuario}
    - IP: {ip}
```

```
    - Hora: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}
    """
        contrasena = "contraseña"

        msg = MIMEMultipart()
        msg["From"] = remitente
        msg["To"] = receptor
        msg["Subject"] = asunto
        msg.attach(MIMEText(mensaje, "plain"))

        try:
            servidor = smtplib.SMTP("smtp.gmail.com", 587)
            servidor.starttls()
            servidor.login(remitente, contrasena)
            servidor.sendmail(remitente, receptor, msg.as_string())
            servidor.quit()
            print("[EMAIL] Alerta enviada por correo.")
        except Exception as e:
            print(f"[X] Error al enviar correo: {e}")
```

Figura 71: Código monitor_tarjetas.py 2

Crea un correo y lo envía a una dirección de correo preconfigurada (en este caso, se envía a sí mismo) mediante SMTP de Gmail. Usa una contraseña de aplicación (la contraseña mostrada es ficticia) para autenticarse. El contenido del correo incluye el usuario que accedió, la IP de origen y la hora de acceso. Para que esto funcione, hay que habilitar el acceso con "contraseñas de aplicación" en Gmail.

```
    def bloquear_archivo():
        try:
            os.chmod(ARCHIVO, 0o000)
```

Figura 72: Código monitor_tarjetas.py 3

Esta función revoca todos los permisos del archivo honeypot para evitar accesos posteriores. El modo 0o000 impide cualquier lectura, escritura o ejecución.

```
    def monitorear_accesos():
```

Figura 73: Código monitor_tarjetas.py 4

Es la función principal y monitorea continuamente (cada 2 segundos) los intentos de lectura al archivo. Utiliza ausearch para consultar logs del sistema (proporcionados por auditd). Si detecta un intento nuevo (que no está en eventos_detectados) extrae el UID y lo convierte en nombre de usuario, extrae la IP del evento, muestra un mensaje de alerta y llama a las funciones registrar_log(), bloquear_archivo(), y enviar_alerta_gmail().

De forma más concreta:

```
      resultado = subprocess.run(
                        ["ausearch", "-k", CLAVE, "-ts", timestamp, "--
format", "raw"],
                        stdout=subprocess.PIPE
                    )
```

Figura 74: Código monitor_tarjetas.py 5

Ejecuta el comando "ausearch" para buscar eventos desde cierta hora (timestamp) con la clave "acceso_tarjetas".

```
      uid_line = next((line for line in log.splitlines() if "uid=" in
line and "auid=" in line), None)
```

Figura 75: Código monitor_tarjetas.py 6

Busca la línea donde aparece el UID real (usuario del sistema que accedió).

```
       usuario = pwd.getpwuid(uid).pw_name
```

Figura 76: Código monitor_tarjetas.py 7

Convierte el UID en nombre de usuario.

```
      addr_line = next((line for line in log.splitlines() if "addr="
in line), None)
      ip = addr_line.split("addr=")[-1].strip().split()[0]
```

Figura 77: Código monitor_tarjetas.py 8

Extrae la IP desde la línea que contiene addr=. Para que la herramienta se ejecute de forma automática al iniciar el sistema, se configura como un servicio de systemd. El archivo my_herramienta.service se copia en la ruta /etc/systemd/system/monitor_tarjetas.service, y contiene los parámetros necesarios para que el sistema gestione su ejecución como un demonio permanente. A continuación, se muestra el contenido presente en el archivo previamente mencionado.

```
[Unit]
Description=Monitor de accesos a archivo sensible
After=network.target auditd.service

[Service]
Type=simple
ExecStart=/usr/bin/python3 /opt/monitor_archivo/monitor_tarjetas.py
Restart=on-failure
User=root
Group=root

[Install]
WantedBy=multi-user.target
```

Figura 78: Código monitor_tarjetas.py 9

En resumen, esta herramienta simula un escenario realista de ataque mediante un honeypot. Permite detectar intentos de acceso no autorizados y actuar de forma automática, alertando al administrador y restringiendo futuras acciones. Su integración con auditd y el sistema de logs del kernel garantiza una monitorización precisa y eficiente.

4.2.3 Aproximaciones fallidas

El desarrollo de la herramienta no estuvo exento de contratiempos y soluciones fallidas que debieron ser replanteadas. Se enumeran a continuación seis aproximaciones o intentos problemáticos que surgieron durante su implementación, describiendo qué se

buscaba lograr, por qué no funcionó como se esperaba y cómo se terminó resolviendo la situación:

1. **Sondeo ingenuo de archivos señuelo:** En la primera aproximación para implementar el honeypot, se usó un método sencillo: comprobar periódicamente la marca de tiempo de último acceso del archivo señuelo mediante llamadas como os.path.getatime(). La expectativa era detectar cualquier acceso no autorizado porque modificaría el timestamp de acceso. Esta técnica falló por varias razones: en sistemas modernos, el *atime* puede no actualizarse por rendimiento; además, un atacante sofisticado podría copiar el archivo sin cambiar su atime (por ejemplo, usando herramientas que evitan tocar esos metadatos). Tras pruebas infructuosas (no se detectaban accesos simulados), se abandonó este enfoque en favor de mecanismos de auditoría más fiables. La solución fue integrar el subsistema auditd para obtener eventos precisos de acceso, en lugar de depender de metadatos de ficheros.
2. **Detección de usuario atacante inexacta:** En una versión intermedia, se detectaba que se accedió al señuelo, pero no conseguía identificar correctamente quién lo había hecho. Inicialmente, se trató de extraer el nombre de usuario del proceso atacante usando el UID registrado en los logs de auditoría. Sin embargo, un error en el parsing (confundir auid con uid en la salida de ausearch) llevó a notificaciones con "Usuario: desconocido". Se intentó paliar asignando temporalmente el usuario como "desconocido" por defecto, pero esto era insatisfactorio para un sistema de alerta. La corrección vino tras depurar el formato de los eventos auditd: se actualizó el código para buscar explícitamente uid= y luego usar la biblioteca pwd de Python para convertir el UID en nombre de usuario. Con esta modificación, las alertas comenzaron a indicar correctamente el nombre de la cuenta involucrada en el acceso.
3. **Alertas repetidas por el mismo evento:** Al probar la herramienta en modo servicio, se observó que generaba múltiples alertas por un único incidente. Esto ocurrió porque el bucle de monitorización consultaba los eventos auditd frecuentemente y, debido a cómo ausearch formatea resultados (separando un mismo evento en múltiples registros o líneas), la lógica inicial interpretaba cada

línea relevante como un evento distinto. El resultado fue que un único acceso al honeypot disparaba varias notificaciones idénticas. La primera solución tentativa fue introducir un *delay* mayor en el ciclo (esperar más tiempo para evitar duplicados), pero no resolvía el problema de fondo. Finalmente, se implementó un mecanismo de de duplicación: se creó una estructura (un conjunto eventos_detectados) para guardar identificadores únicos de eventos ya procesados (combinando usuario, archivo e IP). Además, se ajustó la búsqueda con ausearch para delimitar por tiempo (-ts) de modo que cada consulta solo devolviera eventos nuevos desde la última iteración. Con estas medidas, se eliminó la duplicación de alertas.

4. **Intento fallido de bloqueo activo de intrusos:** En una fase temprana, se consideró que la herramienta no solo alertara, sino que tomara acción al detectar un acceso sospechoso. En primer lugar, se experimentó con que el script ejecutase un comando de sistema como iptables -A INPUT -s <IP_atacante> -j DROP para bloquear la dirección IP de origen si era remota. Esta idea, aunque proactivas, resultó problemática: añadir reglas de firewall sin criterio refinado podía bloquear al propio administrador si se conectaba remotamente durante pruebas. Finalmente, se consiguió que bloquear el acceso al archivo monitorizado mediante el cambio de permisos usando os.chmod(ARCHIVO, 0o000). De esta manera, el atacante solamente podría acceder al archivo una vez.
5. **Configuración de correo electrónico:** Una parte importante de la herramienta es el envío de correos de alerta. Al principio se intentó usar directamente el comando del sistema mail o sendmail para notificaciones, ya que habría simplificado la implementación (sólo llamando a subprocess.run(["mail", "-s", "Alerta", admin_email]...)). Sin embargo, esta aproximación falló en entornos de prueba donde no había un servidor de correo local configurado o las políticas de seguridad bloqueaban el envío externo. Las notificaciones no llegaban o acababan en spam sin rastro claro. La solución fue integrar el envío vía SMTP autenticado usando Python (smtplib con un servidor conocido, Gmail en este caso). Esto requirió aprender a configurar un correo de servicio (con contraseña de aplicación) y manejar posibles excepciones en el envío (ej., falta de

conectividad a Internet, credenciales inválidas). Hubo iteraciones de prueba y error hasta conseguir que los correos se enviaran de forma consistente; en ese proceso, se descubrieron detalles como la necesidad de servidor.starttls() antes del login SMTP. Superados esos obstáculos, la notificación por email quedó funcionando correctamente.

6. **Falsos positivos durante el arranque del sistema:** Al probar la herramienta en arranque automático, se observaron alertas durante los primeros segundos del boot. Esto se debió a que ciertos servicios legítimos accedían al archivo monitorizado durante el inicio. La herramienta, al iniciarse simultáneamente, interpretaba esos accesos normales de arranque como eventos sospechosos. Para solucionar este falso positivo, se implementó en el código una lógica de inicial: se calcula el tiempo de arranque del sistema (mediante timestamp = datetime.now().strftime("%H:%M:%S")) y se ignoran eventos ocurridos con anterioridad a la ejecución de la herramienta. Esta modificación eliminó virtualmente todos los falsos positivos de arranque en las pruebas siguientes, asegurando que solo se atiendan eventos relevantes una vez el sistema está en régimen estable.

4.2.4 Métricas de Complejidad

Para dimensionar el esfuerzo y la complejidad asociados al desarrollo de la herramienta, se pueden analizar varias métricas clave, algunas similares a las de la herramienta anterior pero adaptadas a las particularidades de este proyecto:

- **Número de iteraciones de diseño:** Monitoring_Tool atravesó al menos 3 iteraciones principales en su concepción: (1) versión inicial con monitorización de ficheros simple, (2) versión intermedia integrando auditd y mejorando identificación de eventos, (3) versión final con servicio estable y alertas afinadas. Este número relativamente menor de iteraciones frente a Audit-Tool se debe a que el alcance de Monitoring_Tool era más acotado; aun así, refleja un proceso donde se incorporaron cambios de arquitectura en fases distintas.
- **Intentos fallidos/dificultades encontradas:** Se documentaron 6 aproximaciones fallidas significativas durante el desarrollo (descritas arriba). Este número de

contratiempos es indicativo de la complejidad de la tarea de monitorización en tiempo real: incluso en una herramienta más corta en código, emergieron numerosos desafíos (tecnológicos y lógicos) que requirieron solución. Un alto conteo de dificultades resueltas suele correlacionar con una mayor robustez final, dado que la herramienta ha sido *curtida* por así decirlo, ante múltiples escenarios.

- **Horas de desarrollo:** Se estiman aproximadamente 60-80 horas dedicadas al desarrollo de la herramienta. Aunque el código es menos extenso que el de Audit-Tool, integrar correctamente la herramienta con el sistema (auditd, systemd, correo) demandó investigación y pruebas, lo que consume tiempo. La configuración y testeo en entornos reales (simulando ataques) también sumó a esta inversión temporal. La cantidad de horas refleja el esfuerzo de lidiar con detalles de sistema que van más allá de la lógica del programa en sí.
- **Líneas de código:** La herramienta, en su script principal consolidado, abarca cerca de 150 líneas de código Python, complementadas por configuraciones (un archivo de servicio systemd, reglas de auditd escritas fuera del código). Si bien el número de líneas es modesto comparado con la otra herramienta, conviene notar que la complejidad aquí radica más en la *orquestación con el entorno* que en la longitud del código. De todos modos, esta métrica es útil para dimensionar el tamaño del proyecto: menos líneas implican menos superficie de mantenimiento pero no necesariamente menos complejidad lógica, dado que muchas funciones realizan llamadas al sistema y procesan resultados binarios o textuales.
- **Commits y control de cambios:** El repositorio registró alrededor de 25 commits hasta alcanzar la versión final. Cada commit usualmente correspondió a ajustes finos (por ejemplo, "*Añadir duplicación de eventos*" o "*Ajustar temporización inicial*"). El número de commits, muestra un desarrollo incremental y cuidadoso: se hacían cambios, se probaban, y se consolidaban antes de proceder, lo que es habitual en herramientas que interactúan con el sistema donde un cambio brusco puede romper compatibilidad.
- **Dependencias y entorno de ejecución:** A diferencia de Audit-Tool, esta herramienta depende fuertemente de componentes externos del sistema: auditd

para capturar eventos y un servidor SMTP para enviar correos. Aunque en términos de librerías Python instaladas no sumó muchas, su complejidad se mide también por la necesidad de configurar el entorno. Por ejemplo, sin auditd activo, la mitad de la funcionalidad carece de fuente de datos; sin acceso a Internet o correcta configuración SMTP, las alertas por correo fallan. Estas dependencias del entorno añaden complejidad en la medida en que en el desarrollo debimos contemplar la instalación, configuración y posibles fallos de esos componentes. Se consideró esto en las métricas porque condiciona tanto el desarrollo (pruebas que requieren permisos especiales, entornos con auditd habilitado) como la implementación en producción.

En resumen, las métricas anteriores reflejan que, si bien la herramienta Monitoring_Tool es más pequeña en tamaño que Audit-Tool, su desarrollo involucró retos de integración sistémica y refinamiento lógico. El número de intentos fallidos y horas invertidas valida la importancia de un enfoque meticuloso: una herramienta de seguridad en tiempo real debe ser altamente confiable, lo cual solo se logra tras iterar y pulir muchos detalles. La moderada cantidad de código oculta una complejidad inherente elevada, pues orquestar componentes del sistema operativo para cooperar (auditd, systemd, etc.) es una tarea que va más allá del código mismo y entra en el terreno de la administración de sistemas, incrementando el nivel de dificultad del proyecto.

4.3 Defense-Tool

Esta tercera herramienta implementa un sistema de defensa activa capaz de reaccionar en tiempo real ante accesos no autorizados a recursos críticos del sistema. En nuestro entorno seguro, Defense-Tool supervisa continuamente indicadores de intrusión (sobre todo intentos de lectura en ficheros sensibles como /etc/shadow) y ejecuta automáticamente una serie de contramedidas: registro del evento, envío de alertas por correo, bloqueo de permisos y clasificación del intento (exitoso o fallido).

Para garantizar su persistencia y ejecución autónoma, la herramienta se instala como un servicio systemd (defense.service), de modo que arranca junto al sistema y permanece activo en segundo plano sin intervención manual. A continuación,

describiremos paso a paso su desarrollo, los comandos empleados, los retos superados y las métricas que ilustran su complejidad.

4.3.1 Proceso de desarrollo

La herramienta Defense-Tool surgió con el objetivo de complementar a las anteriores herramientas proporcionando una capa defensiva activa. Tras la experiencia con las anteriores herramientas, se decidió que la herramienta Defense-Tool sería implementada como un demonio residente, ejecutándose desde el arranque del sistema y vigilando continuamente ciertos eventos críticos. Se creó un nuevo repositorio Git (git init) para esta herramienta, con commits iniciales para configurar la estructura: un script principal (defense_system.py) y un directorio para logs generados que se descartaría posteriormente. Dado su propósito, desde el inicio se previó la necesidad de integrarse con componentes de bajo nivel (como auditd nuevamente, o utilidades de red), por lo que el README incluyó instrucciones sobre requisitos del sistema (tener auditd activo, permitir envío de correos, etc.).

En las etapas iniciales, se definieron las superficies que la herramienta protegería. Se creó una lista de archivos críticos a vigilar, que por defecto incluyó /etc/shadow (archivo de contraseñas del sistema, cuyo acceso ilegítimo indica una grave intrusión). Esta lista era fácilmente extensible para agregar otros archivos sensibles si se deseaba, cómo el accedo a /etc/shadow o a otros achivos críticos del sistema. También se establecieron parámetros de funcionamiento como el intervalo de monitorización (pcomprobar eventos cada 2 segundos) y una clave de auditoría única (clave_defensa) para etiquetar los eventos capturados por auditd relativos a esta herramienta. Estas constantes se definieron al inicio del script defense_system.py para facilitar ajustes.

La herramienta adoptó desde el principio una estrategia basada en el subsistema de auditoría del kernel, similar a la herramienta Monitoring_Tool pero con un ámbito más automatizado. En concreto, la lógica principal consiste en un bucle infinito que consulta periódicamente los registros de auditoría en busca de eventos relacionados con los archivos críticos definidos. Se utilizó intensivamente la herramienta ausearch (parte de auditd) con filtros precisos: la clave de auditoría (-k clave_defensa) y un timestamp para obtener solo eventos recientes. Este bucle se implementó de forma resiliente: se

agregaron retardos leves y manejo de excepciones para asegurar que el demonio no consumiera CPU innecesariamente ni se detuviera ante condiciones excepcionales (por ejemplo, si auditd tardaba en responder).

Una de las primeras versiones de este monitor activo simplemente registraba los eventos detectados en un archivo de log (logs/accesos.log), anotando datos como usuario, IP de origen (si figuraba en el evento, lo cual ocurre por ejemplo si la acción proviene de una sesión remota via SSH) y tipo de acceso (éxito o fallo). Esto ya proporcionaba un histórico útil, pero no generaba alertas inmediatas ni tomaba acciones.

Evolucionando el prototipo, se incorporaron notificaciones en tiempo real por correo electrónico. Cada vez que se detecta un evento sospechoso (un usuario accede a /etc/shadow), la herramienta construye un correo de alerta con los detalles del incidente. Se eligió nuevamente utilizar un servidor SMTP externo (Gmail) para enviar estos correos, reusando la experiencia de configuración obtenida con Monitoring_Tool. En el código, se creó una función dedicada enviar_alerta_gmail(usuario, ip, ruta, estado) que arma el mensaje con formato claro (incluyendo fecha, usuario implicado, IP de origen y el tipo de evento, como *INTENTO EXITOSO* o *INTENTO FALLIDO* de acceso) y lo envía usando smtplib. Esta funcionalidad implicó integrar credenciales seguras (contraseña de aplicación) y fue probada en entorno real con un correo de pruebas.

Además de notificar, se implementó una contramedida activa básica: la protección temporal del recurso atacado. Por ejemplo, si detecta un acceso no autorizado a /etc/shadow, inmediatamente ejecuta os.chmod("/etc/shadow", 0o000) para revocar todos los permisos del archivo, impidiendo lecturas posteriores mientras se investiga el incidente. Esta acción de bloqueo es reversible manualmente por el administrador (restaurando permisos adecuados), pero su presencia añade una capa de defensa: limita la ventana de tiempo en la que un atacante puede explotar el acceso a ese archivo. La decisión de incluir esta contramedida se tomó tras deliberar sobre los pros y contras (como se discutió en las aproximaciones fallidas) y se implementó con cuidado para evitar efectos colaterales: por ejemplo, se aseguró que la herramienta no bloqueara repetidamente un archivo ya bloqueado, y que registrara esta acción en el log para que el administrador sepa que ocurrió.

Esta herramienta, al igual que Monitoring_Tool, fue diseñada para correr como servicio del sistema. Se creó el archivo defense_tool.service con las configuraciones necesarias (ejecución como root, inicio pronto en el boot). Sin embargo, se ajustó intencionalmente la secuencia de arranque: dado que esta herramienta debe ignorar eventos del boot (que podrían incluir accesos legítimos a archivos críticos por parte de scripts de inicio), se programó un retardo interno. En el código, se obtiene el timestamp de arranque del sistema (boot time) y se omiten todos los eventos cuya marca de tiempo sea muy cercana a ese arranque (por ejemplo, se decide ignorar los eventos ocurridos en los primeros 30 segundos de uptime). Adicionalmente, la herramienta imprime un mensaje inicial [*] Esperando a que el sistema se estabilice... y duerme unos segundos al comienzo para dar tiempo a terminar el arranque. Esto se hizo para asegurar que el sistema de defensa no reaccione a operaciones normales del sistema durante su fase de inicio.

En la versión final de la herramienta, se combinaron todas las piezas: monitorización auditd, notificaciones por correo, logging local y bloqueo preventivo. Se llevaron a cabo pruebas de estrés para evaluar su comportamiento. Por ejemplo, se simuló un ataque intentando leer /etc/shadow varias veces rápidamente, verificando que la herramienta:

- Detectaba el primer acceso, enviaba un correo de alerta inmediatamente con los detalles correctos.
- Aplicaba el bloqueo de permisos una vez (de modo que intentos subsiguientes fallaban por falta de permisos, limitando daño).
- No inundaba con correos repetidos gracias al mecanismo de registro de eventos ya atendidos.
- Mantenía su ciclo corriendo establemente sin consumir CPU excesiva.
- Podía recuperarse de una pausa/resume (por ejemplo, si el servicio se reiniciaba, solo notificaba nuevos eventos a partir de ese reinicio, evitando duplicados de lo ya registrado antes).

Las pruebas confirmaron la robustez de la herramienta: en entornos controlados, la herramienta reaccionó de forma oportuna y controlada, cumpliendo su objetivo de mitigar en tiempo real ciertos incidentes críticos.

Finalmente, hemos incluido un documento como Apéndice G en el que se describe en profundidad la implementación de la herramienta. Este apéndice recoge el diseño de la arquitectura interna y la explicación completa de los códigos y sus funciones.

4.3.2 Estructura y Arquitectura de la herramienta

- **defense_system.py**

Programa principal de la herramienta donde se desarrolla la lógica completa de la herramienta. Este script monitorea intentos de lectura sobre archivos sensibles, en este caso /etc/shadow como ejemplo, para mantener el entorno seguro. Si detecta que alguien intenta leerlo,r egistra el evento en un archivo de log, envía un correo de alerta, bloquea el archivo (quita todos los permisos),clasifica el intento de acceso en exitoso o fallido y finalmente guarda eventos ya procesados.

```
import subprocess
import os
import time
from datetime import datetime
```

Figura 79: Biblioteca defense_system.py 1

Para control del sistema, tiempo y ejecución de comandos de shell (como ausearch).

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
```

Figura 80: Biblioteca defense_system.py 2

Para el envío de correos, clave para el envío de la alerta en tiempo real.

```
import pwd
```

Figura 81: Biblioteca defense_system.py 3

Traducir UID (User Id) a nombre de usuario.

```
import psutil
```

Figura 82: Biblioteca defense_system.py 4

Obtención del tiempo de arranque del sistema. Para la configuración inicial del programa, necesitamos definir las rutas de los archivos sensibles a monitorear, por ejemplo "/etc/shadow", la clave del correo electrónico utilizada por auditd para identificar los eventos "clave_defensa", el intervalo de medida (2 segundos) y una variable para que se espere 30 segundos desde el arranque del sistema. De esta manera es muy fácil añadir más archivos sensibles a monitorear, tan solo habría que definir su ruta y configurarlo en su instalación.

Posteriormente hay que registrar los eventos en "logs/accesos.log", incluyendo hora, usuario e IP de intento. También hay que guardar las alertas enviadas para que no se repitan en la variable eventos_detectados.

```
    def registrar_log(usuario, ip, ruta, estado):
```

Figura 83: Código defense_system.py 1

Como en monitoring_tool, esta función se encarga de registrar los logs en un formato legible indicando el usuario que intentó acceder y desde que ip, añadiendo la ruta a la que intentó acceder y el estado (si lo consiguió o no). Estas nuevas funcionalidades permiten diferenciar cuál fue el archivo sensible al que se atacó en caso de que haya más añadidos a la ruta inicial.

```
    def enviar_alerta_gmail(usuario, ip, ruta, estado):
```

Figura 84: Código defense_system.py 2

Como en monitoring_tool, es la función encargada de enviar la alerta cuando se produce un acceso sospechoso. Para ello, registra el usuario, su ip, la ruta accedida y si pudo acceder o no. Hay que crear una nueva contraseña de aplicación para diferenciarla de la anterior herramienta. El resto del funcionamiento del código es el mismo.

```
    def proteger_archivo(ruta):
        try:
            os.chmod(ruta, 0o000)
```

Figura 85: Código defense_system.py 3

También cambia los permisos del archivo a 000, es decir, nadie puede leer ni escribir el archivo. Esto es una medida temporal hasta que el administrador actúe.

```
    def obtener_tiempo_arranque():
        """Obtiene el tiempo de arranque del sistema en segundos."""
        boot_time = psutil.boot_time()
        return boot_time
```

Figura 86: Código defense_system.py 4

Detecta la hora a la que el sistema se encendió, útil para evitar alertas falsas inmediatamente después del reinicio. Su implementación nace de la necesidad de reforzar la fiabilidad de la herramienta, ya que se generaban alertas falsas en el arranque del sistema.

```
    def monitorear()
```

Figura 87: Código defense_system.py 5

La función principal del programa que explicaremos paso a paso comienza esperando 10 segundos, lo que permite que el sistema se estabilice tras un posible arranque o reinicio. Luego, define un timestamp (hora actual) que servirá como punto de referencia para ignorar eventos anteriores al inicio del monitoreo. A partir de ahí, cada 2 segundos ejecuta el comando "ausearch -k clave_defensa -ts <timestamp> --format raw", que consulta los eventos recientes registrados por auditd relacionados con accesos a archivos sensibles.

Por cada log detectado, el programa extrae información clave como el nombre del usuario (a partir del UID), la dirección IP de origen (si está disponible), y determina si el intento de acceso fue exitoso o fallido. Antes de procesar un evento, comprueba que no se trate de un duplicado y también descarta aquellos que ocurren muy poco tiempo después del arranque, para evitar falsos positivos.

Si el evento es nuevo y válido, se registra en un archivo de log, se protege inmediatamente el archivo afectado retirando sus permisos, se envía una alerta por correo electrónico al administrador y finalmente se marca como procesado para evitar

repetir acciones sobre el mismo intento. Esta lógica permite una vigilancia constante, efectiva y automática sobre archivos críticos del sistema.

Muestra mensajes en consola indicando que el sistema está empezando a monitorizar. Espera 10 segundos para permitir que el sistema "se estabilice", útil tras el arranque o reinicio.

```
    tiempo_arranque = obtener_tiempo_arranque()
    timestamp = datetime.now().strftime("%H:%M:%S")
    print(f"[*] Ignorando eventos anteriores a {timestamp}\n")
    time.sleep(1.2)
```

Figura 88: Código defense_system.py 6

obtener_tiempo_arranque() guarda el tiempo actual en formato Unix (time.time()). Esto permite ignorar eventos ocurridos muy cerca del arranque del script, evitando falsos positivos. timestamp en formato HH:MM:SS se usa para filtrar eventos auditados más recientes desde que empezó el monitoreo. Con time-sleep(1.2) se genera una pausa corta para dar tiempo a que los eventos empiecen a generarse y evitar sobrecargar el sistema.

```
    while True:
                resultado = subprocess.run(
                    ["ausearch", "-k", CLAVE, "-ts", timestamp, "--
format", "raw"],
                    stdout=subprocess.PIPE
                )
                logs = resultado.stdout.decode().split("\n\n")
```

Figura 89: Código defense_system.py 7

Para detectar accesos sospechosos, el programa ejecuta periódicamente el comando ausearch, una herramienta de auditoría del sistema. Utiliza la opción -k junto a una clave específica previamente definida en una regla de auditd, lo que permite filtrar únicamente los eventos relevantes relacionados con intentos de acceso a archivos sensibles. La opción -ts indica el momento a partir del cual deben considerarse los eventos, es decir, desde que el script comenzó a ejecutarse, ignorando cualquier

actividad anterior. Finalmente, la salida se genera en formato raw gracias a la opción --format raw, lo que facilita su análisis automático línea por línea dentro del programa.

```
      for log in logs:
            if log and log not in eventos_detectados:
```

Figura 90: Código defense_system.py 8

Recorre cada log (evento del auditd) y si el log no está vacío y no ha sido ya procesado (eventos_detectados evita duplicados), continúa.

```
      if "name=" in line:
             for archivo in ARCHIVOS_PELIGROSOS:
                    if f'name="{archivo}"' in line:
                        ruta = archivo
                        break
```

Figura 91: Código defense_system.py 9

Busca líneas que contengan el nombre de un archivo (name="...") y si coincide con alguno en la lista ARCHIVOS_PELIGROSOS, guarda esa ruta como la accedida.

```
      if "type=SYSCALL" in line and "success=" in line:
                                  partes = line.split()
                                  success_val = next((p for p in partes if
p.startswith("success=")), "")
                                  estado = "INTENTO EXITOSO" if success_val
== "success=yes" else "INTENTO FALLIDO"
```

Figura 92: Código defense_system.py 10

Busca eventos SYSCALL para determinar si el intento fue exitoso o fallido (success=yes o success=no).

```
      if "uid=" in line and "auid=" in line:
                                  try:
                                      partes = line.split()
                                      for parte in partes:
                                          if parte.startswith("uid="):
```

```
                                                uid                                =
int(parte.split("=")[1])
                                                usuario                            =
pwd.getpwuid(uid).pw_name
                                        except:
                                            pass
```

Figura 93: Código defense_system.py 11

Extrae uid= (ID de usuario real) de la línea. Usa pwd.getpwuid(uid).pw_name para obtener el nombre del usuario asociado.

```
      if "addr=" in line:
          ip = line.split("addr=")[-1].split()[0]
```

Figura 94: Código defense_system.py 12

Si hay una dirección IP (addr=), la extrae.

```
      tiempo_actual = time.time()
                            if    tiempo_actual    -    tiempo_arranque    <
ARRANQUE_TEMPRANO_SEGUNDOS:
                                print(f"[IGNORADO]    Acceso    durante    el
arranque: {usuario} | Ruta: \"{ruta}\"")
      continue
```

Figura 95: Código defense_system.py 13

Se calcula el tiempo transcurrido desde que el sistema arrancó y si es menor que el umbral definido (30 segundos), se ignora el evento. Esto evita falsos positivos por procesos que acceden a archivos justo al inicio del sistema.

```
      evento_id = f"{usuario}-{ruta}-{ip}"
      if evento_id in eventos_detectados:
             continue
```

Figura 96: Código defense_system.py 14

Se construye una "firma" del evento (usuario, ruta, ip) y si ya fue detectado anteriormente, se omite.

```
    print(f"[{estado}] Usuario: {usuario} | Ruta: \"{ruta}\"")
    registrar_log(usuario, ip, ruta, estado)
    proteger_archivo(ruta)
    enviar_alerta_gmail(usuario, ip, ruta, estado)
    eventos_detectados.add(evento_id)
```

Figura 97: Código defense_system.py 15

Si pasa los filtros anteriores, se imprime en consola el acceso detectado y se registra el evento. Después se ejecuta la función para proteger el archivo accedido y se envía la correspondiente alerta al correo con el usuario, su id, la ruta accedida y si pudo o no hacerlo. Por último, se guarda el evento para evitar futuros duplicados.

- **defense.service**

El archivo /etc/systemd/system/defense.service convierte defense_tool en un servicio del sistema, ejecutándose automáticamente al inicio y manteniéndose activo en segundo plano. El contenido del archivo es:

```
    [Unit]
    Description=Defensa de archivos sensibles.
    After=network.target auditd.service
    [Service]
    Type=simple
    ExecStart=/usr/bin/python3 /opt/defense_archivo/defense_system.py
    Restart=on-failure
    User=root
    Group=root

    [Install]
    WantedBy=multi-user.target
```

Figura 98: Archivo defense.service

En resumen, es una herramienta de defensa proactiva que protege archivos críticos del sistema ante accesos sospechosos o no autorizados. Mediante la integración con auditd y el análisis en tiempo real de eventos del kernel, permite detectar accesos a archivos sensibles, clasificarlos (éxito o fallo), registrar los eventos, alertar al administrador por

correo y aplicar medidas inmediatas como el bloqueo de archivos. Su funcionamiento continuo como servicio del sistema, junto con un sistema de registros y alertas, garantiza una vigilancia constante, autónoma y eficaz. Es una solución realista y práctica para reforzar la seguridad de sistemas Linux en tiempo de ejecución.

4.3.3 Aproximaciones fallidas

El desarrollo de la herramienta presentó varios desafíos únicos, dada su intención de actuar automáticamente ante posibles ataques. A través de estas situaciones, el desarrollo fue refinándose hasta lograr una herramienta eficaz. Cada obstáculo resaltó aspectos críticos de diseñar sistemas de respuesta automática: la necesidad de controlar recursos, evitar bucles intensivos, garantizar la precisión en la detección y ser cauteloso al tomar medidas defensivas. A continuación, se detallan seis aproximaciones fallidas o problemas críticos enfrentados durante su creación, junto con sus soluciones finales:

1. **Exceso de consumo de CPU en lazo de monitorización:** La primera implementación del bucle de vigilancia auditd carecía de pausas adecuadas. Se había programado para iterar continuamente (consulta de eventos, análisis, repeat) creyendo que el intervalo de 2 segundos establecido con time.sleep() sería suficiente. En pruebas prolongadas, se detectó un consumo de CPU inusualmente alto; al depurar, se encontró que bajo ciertas condiciones (como ausencia de eventos), la instrucción ausearch terminaba muy rápido y el bucle re-evaluaba casi de inmediato sin respetar totalmente el intervalo. Se intentó inicialmente alargar el sleep, pero eso podía hacer la reacción demasiado lenta. También se probó reducir la prioridad del proceso (nice) para mitigar impacto, sin mucho éxito. La solución fue ajustar la lógica para esperar de manera dinámica: tras procesar eventos, la herramienta calcula el tiempo que resta hasta completar los 2 segundos de ciclo y duerme exactamente ese lapso. Con estas medidas, el uso de CPU se normalizó sin sacrificar la capacidad de respuesta.
2. **Parseo incorrecto de logs de auditoría complejos:** Los eventos capturados de auditd son técnicamente complejos, a veces divididos en múltiples registros (SYSCALL, PATH, etc.). En un intento inicial de simplificar, el código buscaba solo

líneas que contuvieran la ruta del archivo vigilado (name="/etc/shadow") para disparar la alerta, ignorando el resto. Esto funcionaba para accesos exitosos, pero falló en detectar intentos fallidos donde quizás la ruta no aparecía explícitamente o venía en un bloque separado. Como consecuencia, la herramienta inicialmente no alertaba de intentos fallidos de abrir /etc/shadow (lo cual es igualmente importante de saber). Al descubrir esta laguna, se intentó remediar concatenando todos los fragmentos de log en uno y buscando en ese texto combinado. Sin embargo, esto generó falsos positivos y dificultades en extraer quién/qué hizo qué. La solución llegó al implementar un parser más granular: en lugar de usar cadenas simples, se recorren las líneas identificando campos claves (uid, success, name, addr). Se tuvo que iterar varias veces sobre esta lógica hasta cubrir todos los casos: por ejemplo, inicialmente no se manejaba el caso en que addr= no aparece (indicando un proceso local), lo que resultaba en un ip "localhost" por defecto. Tras ajustar para contemplar tanto eventos locales como remotos y tanto éxitos como fallos, el parseo se volvió confiable.

3. **Notificaciones de correo no enviadas (silenciosamente):** Al integrar el envío de emails, se asumió que cualquier error lanzaría una excepción visible. Sin embargo, durante una prueba de desconexión de red, la herramienta detectó un evento y aparentemente continuó sin problemas, pero el correo nunca llegó. Esto evidenció que ciertas excepciones en smtplib podían estar siendo capturadas inadvertidamente o no registradas. Inicialmente se agregó un print de depuración para cualquier excepción de envío, pero como era un servicio, ese print solo aparecía en logs, pasando desapercibido. Esta situación se interpretó erróneamente como correos "perdidos". La solución fue mejorar la gestión de errores y logging: se envolvió el bloque de envío de correo en un try/except específico que captura cualquier smtplib.SMTPException o error de red, y en caso de ocurrir, registra en el log local un mensaje de error (prefijado con [X]). De esta forma, si una alerta no pudo enviarse, queda constancia para su posterior análisis. Adicionalmente, se implementó una política simple de reintento: si falla el envío, la herramienta puede intentar una vez más tras un breve retraso, para evitar perder una alerta crítica por un fallo transitorio de red.

4. **Demasiados eventos auditd por una sola acción:** En ciertas ocasiones, una única acción maliciosa generaba múltiples eventos en auditd. Por ejemplo, un script malicioso que leía y luego escribía en /etc/shadow (para alterar contraseñas) produciría tanto un evento de lectura como otro de modificación. La herramienta, al principio, reaccionaba a ambos por separado: enviaba dos correos, registraba dos entradas y ejecutaba chmod dos veces (la segunda innecesaria). Esto saturaba un poco la respuesta. Se exploró la idea de combinar eventos cercanos en el tiempo en uno solo, pero resultaba complejo correlacionar automáticamente. En su lugar, se implementó un filtro por evento único mediante la identificación compuesta (usuario-ruta-ip ya se usaba) más el tipo de acción. Es decir, se genera un ID interno del evento incluyendo si fue éxito o fallo. Así, dos eventos sobre la misma ruta por el mismo usuario, pero con diferente permiso (r/w) se consideran distintos solo si la acción difiere. Aun así, para reducir ruido, se optó por centrarse en accesos de lectura no autorizados principalmente. Esta priorización y filtrado redujo efectivamente la duplicación de respuestas.
5. **Conflicto con otros sistemas de seguridad:** Durante la integración se descubrió que el uso de auditctl para /etc/shadow podía interferir con herramientas de seguridad que probamos ya existentes (por ejemplo, SELinux o AppArmor generando sus propios logs, o un gestor de contraseñas que legítimamente toca /etc/shadow). Inicialmente, esto se interpretó como un fallo de la herramienta, pero era una interacción con otra capa de seguridad. Este suceso mostró la importancia de convivir con otros mecanismos de seguridad sin generar efectos adversos.
6. **Gestión de estados tras reinicio del servicio:** Se notó en pruebas que si la herramienta se reiniciaba (por ejemplo, el proceso se detenía y volvía a iniciar manualmente), podría reenviar alertas de eventos antiguos no purgados de los logs de auditd, interpretándolos como nuevos. Esto sucedía porque ausearch -ts <hora_inicio> empezaba de cero en cada arranque del servicio. Se intentó mantener un registro persistente del último evento procesado (p. ej., guardando en un archivo el timestamp del último evento manejado antes de reiniciar), pero

se encontró que conciliar el tiempo exacto entre reinicios era problemático, especialmente si el sistema reloj cambiaba o si los logs rotaban. Como solución práctica, se implementó que, al iniciar, la herramienta use por defecto -ts now (es decir, ignore eventos anteriores al momento de arranque del demonio) a menos que se especifique lo contrario. Esto asegura que un reinicio limpio no reprocesa eventos viejos. Si se pierde alguno por reiniciar precisamente en ese instante, se asumió aceptable dado que normalmente el servicio está siempre activo. Se agregó esta consideración en el código y comentarios para claridad.

4.3.4 Métricas de Complejidad

Finalmente, se evalúa la complejidad del desarrollo de Defense-Tool mediante métricas y medidas significativas, para entender el esfuerzo invertido y los desafíos superados:

- **Número de iteraciones de diseño:** La herramienta pasó por 3 iteraciones mayores durante su desarrollo: un prototipo básico de registro de eventos, una versión intermedia con notificaciones y mejoras de parseo, y la versión final con bloqueo activo y ajustes de rendimiento. Cada iteración añadió capacidades importantes (p.ej., de registro a alerta, luego a respuesta activa), implicando reestructuraciones notables del código. Este número de iteraciones indica un proceso de refinamiento progresivo, necesario debido a la ambición de la herramienta de ir más allá de solo monitorear, llegando a intervenir en el sistema.
- **Problemas críticos resueltos:** Se enumeraron 6 problemas/aproximaciones fallidas principales enfrentados en el desarrollo. Este conteo subraya la complejidad del proyecto: incluso con una base de código relativamente pequeña, hacer una herramienta reactiva y fiable requirió abordar numerosos detalles (desde parseo de logs hasta sincronización con otros sistemas de seguridad). La cantidad de problemas superados es un indicador de la profundidad técnica abordada; cada problema demandó análisis y pruebas, reflejando un considerable esfuerzo de desarrollo.
- **Horas de desarrollo:** Se estiman en 80-100 horas las dedicadas a concebir, implementar y probar Defense-Tool. Aunque la herramienta comparte algunos conceptos con Monitoring_Tool, las funcionalidades adicionales (acción

defensiva, mayor robustez en notificaciones) añadieron trabajo extra de diseño y prueba. El tiempo invertido también incluyó experimentación con monitoreo de red que luego se descartó, lo que forma parte del cómputo. Esta cifra de horas es coherente con la complejidad de coordinar múltiples elementos del sistema y garantizar la estabilidad de un demonio de seguridad.

- **Líneas de código:** El script principal defense_system.py de la herramienta cuenta con aproximadamente 160 líneas de código. Adicionalmente, incluye unas 50-60 líneas de comentarios y constantes configurables, y genera un log externo. La cantidad de código es menor que la de las otras herramientas, pero esto es engañoso respecto a complejidad: la densidad de lógica por línea es alta (muchas líneas realizan llamadas al sistema o procesan strings significativas). La brevedad en tamaño del código fue posible gracias a reutilizar componentes del sistema (auditd) en lugar de implementar monitoreos manuales extensos, haciendo que la herramienta fuese lo más eficiente y fiable posible.
- **Commits realizados:** El desarrollo quedó reflejado en unos 57 commits en su repositorio. Este número es superior a la de las otras dos herramientas, ya que el desarrollo requirió más pruebas y muestra que se hicieron bastantes ajustes incrementales. Una cantidad alta de commits sugiere que el código fue evolucionando con atención a detalles, refinando el comportamiento en muchos pequeños pasos, lo cual es característico de proyectos donde la confiabilidad es crucial.
- **Dependencias externas y entorno:** Esta herramienta, al igual que Monitoring_Tool, depende de elementos del entorno más que de librerías Python. Por tanto, la complejidad de entorno es una métrica importante: necesitar coordinar componentes del OS (auditd, systemd) y recursos externos (correo) eleva la complejidad frente a un script aislado. Se valora que la instalación involucra varios pasos (instalar auditd si no está, configurar reglas, obtener credenciales de correo), lo cual se tradujo en mayor documentación y casos de prueba. Esta dependencia de entorno es un factor a tener en cuenta al medir la complejidad global del desarrollo, pues el desarrollo requirió tener conocimientos más allá de

Python, incluyendo administración de sistemas Linux y seguridad de la información.

- **Tiempo de respuesta y cobertura de protección:** Una métrica de desempeño relevante es el tiempo de respuesta a incidentes, que se logró mantener en alrededor de 2-3 segundos desde la ocurrencia de un evento hasta la acción (alerta/bloqueo). Conseguir este tiempo acotado requirió optimizaciones en el código (ciclo frecuente pero ligero) y es indicativo de la eficacia de la herramienta. Asimismo, la cobertura de protección puede medirse en función de qué recursos vigila: por defecto, un archivo crítico (/etc/shadow), extensible a más. Aunque el número es pequeño, el impacto de proteger ese recurso es muy alto (es el corazón de las credenciales). Este enfoque focalizado mantuvo la herramienta manejable en complejidad: en lugar de intentar cubrir todo el espectro de intrusiones, se centró en indicadores de alto valor. Esta decisión estratégica es difícil de cuantificar, pero es relevante al evaluar la complejidad porque delimita el problema; sin esa delimitación, el proyecto podría haberse vuelto inabarcable.

Considerando todas estas métricas, se puede concluir que el desarrollo de la herramienta Defense-Tool fue complejo en un sentido distinto a Audit-Tool: implicó menos código, pero más ingeniería de integración y fine-tuning del comportamiento. Las horas invertidas y los múltiples intentos reflejan el desafío de crear un sistema de defensa automático fiable. La recompensa de ese esfuerzo es una herramienta que añade una capa crucial de seguridad al entorno: no solo detecta, sino que también reacciona, brindando al administrador del sistema valiosos segundos y conocimiento para contrarrestar una intrusión en progreso. En términos globales, las tres herramientas, cada una con su propio grado de complejidad y especialización, trabajan en conjunto para reforzar la seguridad del entorno Linux, cumpliendo con los objetivos planteados en el proyecto.

Capítulo 5 - Diseño del Entorno

Tal y como aparece en el título de este trabajo, el objetivo principal de este trabajo es la creación de un entorno que permita la celebración de hackatones de ciberseguridad de una manera segura en los laboratorios de la Universidad Complutense de Madrid.

Entender el concepto de "entorno" es clave a la hora de continuar con este capítulo. Tal y como lo hemos planteado en este proyecto, un entorno comprende desde el sistema operativo y sus cambios, pasando por las herramientas implementadas, siguiendo por los documentos proporcionados para llevar a cabo, en este caso, el hackatón.

Como nuestra intención es desarrollar el entorno para celebrar cualquier hackatón en la facultad de informática de la UCM, en este capítulo nos hemos centrado exclusivamente en el diseño del software ya que la documentación proporcionada en el hackatón es única en cada evento, sin embargo, hemos decidido incluir un ejemplo práctico de cómo hemos diseñado un entorno específico para un hackatón, para poder usarlo como entorno de pruebas.

Para abordar este tema, decidimos partir de la base que es el sistema operativo. Una vez establecido el punto de partida, proseguimos con la instalación de herramientas que consideramos necesarias para llevar a cabo estos hackatones, que en nuestro caso son las tres herramientas que hemos desarrollado a lo largo de este proyecto. Finalmente, los cambios aplicados al sistema operativo son también fundamentales para poder llevar a cabo, por ello, indicamos los cambios que hemos considerado pertinentes para ajustar el entorno perfecto para llevar a cabo los hackatones.

5.1 Cambios del Sistema Operativo

Hemos elegido usar el sistema Ubuntu 24.04 ya que, tal y como vimos en el Estado del Arte, es el sistema operativo que usa el laboratorio de la facultad de informática de la Universidad Complutense de Madrid y porque es un sistema operativo LTS, lo cual nos garantiza soporte durante bastante tiempo. La versión de este sistema la hemos obtenido de la página: https://www.osboxes.org/ubuntu/ en donde hemos descargado la versión Ubuntu 24.04 Noble Numbat.

Para implementar correctamente el entorno, tuvimos que hacer una serie de modificaciones al sistema operativo. Estos cambios han consistido en la integración de varias herramientas al sistema y los cambios que conllevaban. En primer lugar, la contraseña para acceder al sistema operativo es *enero2025*, lo mismo que para ejecutar cualquier acción como usuario root.

Las herramientas que hemos desarrollado se han integrado al sistema operativo ya que ambas cumplen los requisitos que establecimos al inicio del proyecto y son necesarias para la realización de los hackatones que se llevarán a cabo.

Los requisitos que necesitan estas herramientas para funcionar están todos indicados en el GitHub de cada herramienta, sin embargo, procedemos a detallarlos. Para ambas herramientas, es necesario contar con Python 3.x.

Para Audit-tool es necesario descargar requests, psutil, pyfiglet, colorama, pip y git.

Para monitoring_tool es necesario descargar pip, systemd y auditd. Además, habría que contar con una cuenta GMAIL para enviar las alertas.

Finalmente, es necesario adaptar nuestro idioma al teclado, cambiando el idioma del inglés al español y también es necesario cambiar la fecha y hora a la zona de Madrid.

Una vez verificados los requisitos previos y con las dependencias correctamente instaladas, se puede proceder a la instalación de las herramientas en el sistema. Aunque toda esta información está también documentada de forma detallada y visual en el repositorio de GitHub del proyecto, a continuación, se describe paso a paso cómo llevar a cabo la instalación dentro del sistema operativo preparado para los hackatones.

5.1.1 Instalación de Audit-Tool

La primera herramienta desarrollada como parte del proyecto tiene como finalidad principal realizar una auditoría completa del sistema operativo, permitiendo la observación de eventos y comportamientos relevantes. Esta herramienta ha sido desarrollada en Python y se ha diseñado para ejecutarse en un entorno Linux, concretamente sobre Ubuntu 24.04 LTS, por su estabilidad, compatibilidad y soporte.

La instalación comienza por la preparación del entorno de trabajo. Para ello, se recomienda utilizar una máquina virtual basada en Ubuntu 24.04, disponible en plataformas como osboxecontraseñas.org, en formato .ova para VirtualBox.

Con el sistema iniciado, el siguiente paso consiste en actualizar los paquetes del sistema ejecutando `sudo apt update && sudo apt upgrade -y`. Posteriormente, se instalan las dependencias esenciales para clonar y ejecutar la herramienta, incluyendo Git, Python 3 y pip, mediante el comando `sudo apt install -y git python3 python3-pip`.

Una vez completadas estas configuraciones básicas, se accede al directorio donde se desea alojar el proyecto, por ejemplo ~/Documents, y se clona el repositorio con el siguiente comando: `git clone https://github.com/fedelm8/TFG`

Con el código descargado, se accede a la carpeta del proyecto (`cd TFG`) y se procede a la creación de un entorno virtual de Python, con el objetivo de aislar las dependencias del sistema. Para ello, se instala el módulo venv mediante `sudo apt install python3-venv`, y se crea el entorno con `python3 -m venv venv`. El entorno se activa con `source venv/bin/actívate`.

Una vez activado el entorno, se instalan las dependencias definidas en el fichero requirements.txt, mediante el comando: `pip install -r requirements.txt`

Este paso asegura que todas las librerías necesarias —como requests, psutil, pyfiglet y colorama— estén disponibles en el entorno virtual de forma controlada.

A continuación, se instalan dos herramientas adicionales de auditoría del sistema, chkrootkit y clamav, que permiten realizar análisis complementarios del estado de seguridad del sistema. Ambas pueden instalarse mediante el gestor de paquetes de Ubuntu ejecutando: `sudo apt install chkrootkit clamav`

Con todas las dependencias configuradas, la herramienta está lista para su ejecución. Para iniciarla, se utiliza el intérprete de Python desde el entorno virtual con privilegios de superusuario, mediante el siguiente comando: `sudo venv/bin/python Audit-Tool.py`

Una vez lanzada, la herramienta realizará un informe del sistema, registrando eventos relevantes y presentando la información en una interfaz clara y visual, adaptada al contexto de un hackatón.

Este proceso de instalación garantiza que la herramienta pueda ejecutarse en un entorno controlado, replicable y funcional, cumpliendo con los requisitos definidos al inicio del proyecto. Gracias al uso de un entorno virtual de Python, la instalación es segura, aislada y fácil de mantener, lo que favorece su uso en diferentes sistemas sin generar conflictos con otras aplicaciones.

5.1.2 Instalación de monitoring_tool

La segunda herramienta desarrollada en este proyecto tiene como objetivo la supervisión en tiempo real del acceso a archivos sensibles del sistema, generando alertas automáticas por correo electrónico ante cualquier intento de lectura no autorizado. Para su correcto funcionamiento, la herramienta se apoya en auditd, el servicio de auditoría del sistema operativo Linux, y se ejecuta como un servicio persistente mediante systemd.

El proceso de instalación comienza con la configuración de auditd. Una vez actualizado el sistema, se procede a la instalación del demonio de auditoría, su activación y arranque. Esta herramienta es la encargada de registrar los eventos del sistema y constituye la base sobre la que opera el script de monitorización. Junto a auditd, también se instalan las herramientas necesarias para clonar el proyecto desde GitHub, así como Python 3 y su gestor de paquetes pip.

Tras instalar las dependencias, se clona el repositorio del proyecto en el sistema, accediendo a la carpeta donde se encuentra el script monitor_tarjetas.py, que será el núcleo funcional de esta herramienta. Para realizar las pruebas iniciales, se simula la existencia de un archivo sensible —por ejemplo, un fichero que contenga números de tarjetas bancarias—, al que se le asignan permisos de lectura estándar. Este archivo será el que posteriormente se monitorice.

Con el archivo preparado, se define una regla de auditoría temporal con auditctl, que permitirá detectar cualquier intento de acceso al mismo. Una vez comprobado que la herramienta registra correctamente los accesos, esta regla se vuelve permanente editando el archivo de configuración de auditd. De este modo, se asegura que la supervisión se mantenga activa incluso tras reiniciar el sistema.

Una característica clave de esta herramienta es su capacidad para enviar notificaciones por correo electrónico ante eventos sospechosos. Para ello, se requiere una cuenta de Gmail configurada con una contraseña de aplicación específica. Esta contraseña, generada desde el panel de seguridad de Google, se introduce directamente en el script para permitir el envío automatizado de mensajes.

Con el script configurado, se procede a asegurar su ejecución en segundo plano convirtiéndolo en un servicio del sistema. Para ello, se traslada a una ubicación protegida, se restringen sus permisos y se define un servicio personalizado de systemd. Este servicio se configura para ejecutarse automáticamente al inicio del sistema, utilizando Python como intérprete, y se establece para reiniciarse en caso de fallo, garantizando así su disponibilidad continua.

Una vez creado el servicio, se recargan las configuraciones de systemd, se habilita el servicio y se inicia su ejecución. Desde este momento, la herramienta queda activa de forma persistente en el sistema, monitorizando los accesos al archivo sensible definido y enviando alertas cuando se detecta un comportamiento anómalo.

Este enfoque, basado en herramientas del propio sistema Linux y sin necesidad de software externo complejo, permite una integración sencilla pero efectiva en entornos orientados a la seguridad como los hackatones. La combinación de auditd, Python y systemd ofrece una solución ligera, fiable y fácilmente desplegable, cumpliendo con los requisitos funcionales y no funcionales establecidos al inicio del proyecto.

5.1.3 Instalación de defense_tool

La tercera herramienta desarrollada en este proyecto tiene como objetivo la supervisión en tiempo real del acceso a archivos sensibles del sistema, generando alertas automáticas por correo electrónico ante cualquier intento de lectura no autorizado. Para su correcto funcionamiento, la herramienta se apoya en auditd, el servicio de auditoría del sistema operativo Linux, y se ejecuta como un servicio persistente mediante systemd.

El proceso de instalación comienza con la configuración de auditd. Una vez actualizado el sistema, se procede a la instalación del demonio de auditoría, su activación y arranque. Esta herramienta es la encargada de registrar los eventos del sistema y

constituye la base sobre la que opera el script de monitorización. Junto a auditd, también se instalan las herramientas necesarias para clonar el proyecto desde GitHub, así como Python 3 y su gestor de paquetes pip.

Tras instalar las dependencias, se clona el repositorio del proyecto en el sistema, accediendo a la carpeta donde se encuentra el script defense_system.py, que será el núcleo funcional de esta herramienta. Para realizar pruebas, se define una regla de auditoría temporal con auditctl, que permitirá detectar cualquier intento de acceso al mismo. Una vez comprobado que la herramienta registra correctamente los accesos, esta regla se vuelve permanente editando el archivo de configuración de auditd. De este modo, se asegura que la supervisión se mantenga activa incluso tras reiniciar el sistema.

Una característica clave de esta herramienta es su capacidad para enviar notificaciones por correo electrónico ante eventos sospechosos. Para ello, se requiere una cuenta de Gmail configurada con una contraseña de aplicación específica. Esta contraseña, generada desde el panel de seguridad de Google, se introduce directamente en el script para permitir el envío automatizado de mensajes.

Con el script configurado, se procede a asegurar su ejecución en segundo plano convirtiéndolo en un servicio del sistema. Para ello, se traslada a una ubicación protegida, se restringen sus permisos y se define un servicio personalizado de systemd. Este servicio se configura para ejecutarse automáticamente al inicio del sistema, utilizando Python como intérprete, y se establece para reiniciarse en caso de fallo, garantizando así su disponibilidad continua.

Una vez creado el servicio, se recargan las configuraciones de systemd, se habilita el servicio y se inicia su ejecución. Desde este momento, la herramienta queda activa de forma persistente en el sistema, monitorizando los accesos al archivo sensible definido y enviando alertas cuando se detecta un comportamiento anómalo.

Este enfoque, basado en herramientas del propio sistema Linux y sin necesidad de software externo complejo, permite una integración sencilla pero efectiva en entornos orientados a la seguridad como los hackatones. La combinación de auditd, Python y

systemd ofrece una solución ligera, fiable y fácilmente desplegable, cumpliendo con los requisitos funcionales y no funcionales establecidos al inicio del proyecto.

5.1.4 Organización en el GitHub

Cada herramienta desarrollada cuenta con su propio repositorio en GitHub. Esta decisión ofrece numerosas ventajas, como una mejor organización del código, una mayor trazabilidad de los cambios y la posibilidad de facilitar el acceso y uso por parte de otros usuarios o desarrolladores. Para estructurar los repositorios, nos hemos inspirado en buenas prácticas observadas en otros proyectos similares alojados en la plataforma.

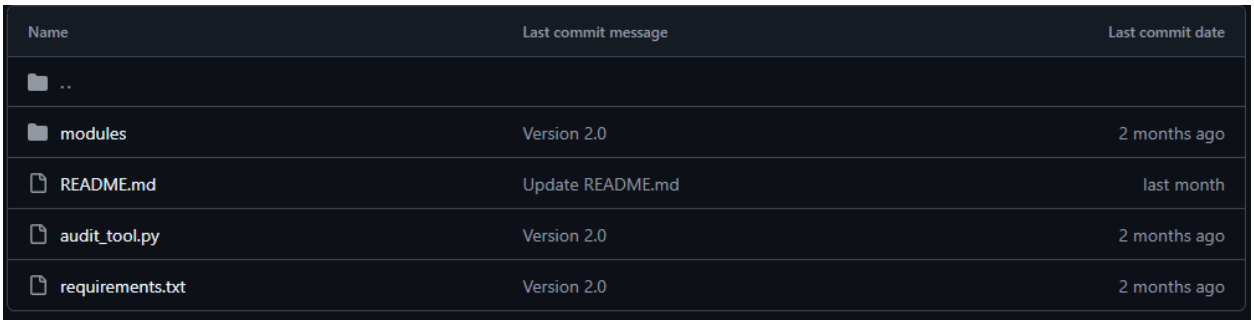

Figura 99: Organización de los repositorios

Como podemos ver en la Figura 99, tenemos un directorio que contiene el código principal que contiene el núcleo del código de la herramienta, acompañado de los módulos auxiliares necesarios. Un archivo de requisitos (requirements.txt) que lista las dependencias necesarias para ejecutar la herramienta correctamente. Un fichero README.md, cuidadosamente redactado, que explica de forma clara y concisa el propósito de la herramienta, las instrucciones de instalación y uso, así como ejemplos prácticos, tal y como vimos en las capturas del capítulo anterior. Esto facilita que cualquier usuario interesado pueda comprender, instalar y utilizar la herramienta sin necesidad de revisar el código fuente en profundidad.

5.2 Entorno de Pruebas

Para comprobar el funcionamiento completo del entorno y detectar posibles mejoras, decidimos ponerlo en práctica preparando el entorno para celebrar un hackatón. Aunque toda la información detallada sobre el evento se incluye en el capítulo

siguiente, en este apartado explicamos los ajustes realizados específicamente a nivel del sistema operativo para adaptar el entorno a esta prueba.

Consideramos esencial demostrar cómo, a partir del entorno base desarrollado, es posible realizar adaptaciones específicas para celebrar hackatones. En nuestro caso, el reto propuesto consistía en **explotar una vulnerabilidad relacionada con el bit SUID (Set User ID)**, el cual permite a un usuario sin privilegios ejecutar un archivo con los permisos de su propietario, pudiendo ser aprovechado para escalar privilegios en el sistema.

Partiendo del entorno que hemos diseñado, hemos añadido una serie de archivos que hemos planeado que sean usados por los participantes para lograr el objetivo del hackatón.

Uno de los archivos que hemos añadido, es un fichero llamado tarjetas_bancarias.txt en la carpeta Documents que contiene lo siguiente:

Lista de tarjetas comprometidas:

**** **** **** 1234

**** **** **** 5678

**** **** **** 5678

**** **** **** 5678

Nota interna: usar clave 'h4ck3r2024' para restaurar backup.

La clave ‘h4ck3r2024’ es necesaria para superar la prueba, ya que se utilizará posteriormente para abrir el archivo protegido del sistema. Para evitar problemas de acceso, asignamos los permisos adecuados mediante el comando chmod. Además, para monitorear los intentos de lectura de este fichero, hemos configurado la herramienta de Monitoring_Tool para que nos avise por email cuando los participantes intenten acceder a él. Para ello, hemos incluido la ruta del fichero tarjetas_bancarias.txt en la variable ARCHIVO al principio del código y hemos cambiado una serie de ajustes para personalizar el email que llegaría como aviso. Además, hemos configurado una cuenta de correo de Google para recibir las alertas ya que usa SMTP de Gmail. Los detalles sobre cómo se configura correctamente la cuenta de correo se encuentra en el capítulo anterior, por lo que no vamos a mencionarlo. El resultado de toda esta

configuración nos acaba proporcionando un resultado parecido a la Figura 100 a continuación.

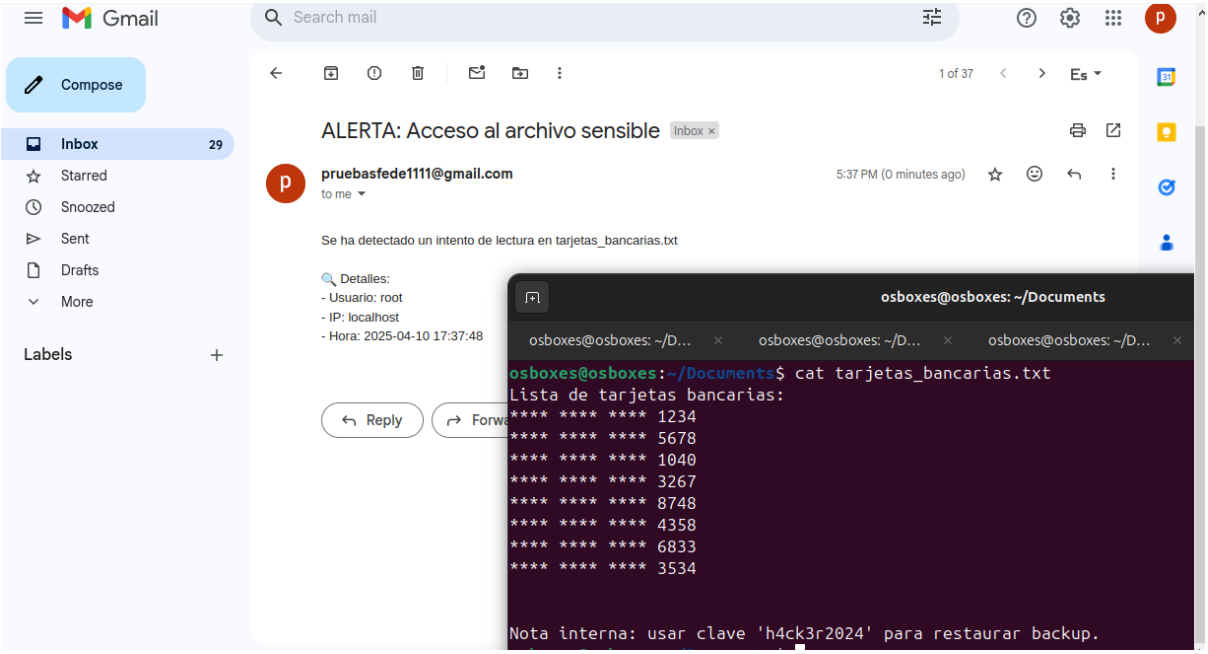

Figura 100: Notificación del HoneyPot

El núcleo del reto se centra en un ejecutable conocido como **"la caja fuerte"**, cuyo propósito es conceder acceso al archivo secretos_del_banco.txt, ubicado en /opt. Este archivo está protegido y solo puede abrirse correctamente a través del ejecutable, previa introducción de la contraseña correcta.

Para aumentar la dificultad, el ejecutable fue ocultado en una ubicación poco habitual del sistema mediante los siguientes comandos:

```
sudo mkdir -p /usr/lib/compatibilidad/.drivers

sudo mv ./caja_fuerte /usr/lib/compatibilidad/.drivers/.sys_check
```

Con esto, se espera que los participantes exploren el sistema en busca de ejecutables con el bit SUID activado, lo cual fomenta habilidades de investigación y análisis del

sistema de ficheros. A continuación, se incluye una captura que muestra el código de "la caja fuerte" (Figura 101):

caja_fuerte.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

void animacion_inicio() {
    printf("\n Iniciando Caja Fuerte Digital");
    for (int i = 0; i < 3; i++) {
        printf(".");
        fflush(stdout);
        sleep(1);
    }
    printf("\n Acceso restringido - se requiere autenticación.\n\n");
}
void acceso_concedido() {
    printf("\n Contraseña correcta.\n");
    printf(" Desbloqueando compartimiento secreto...\n");
    sleep(2);
    printf(" Archivo cargado:\n\n");
}
int main() {
    char pass[64];
    const char *clave = "h4ck3r2024";
    FILE *f;
    animacion_inicio();
    printf(" Introduce la clave secreta para abrir la caja fuerte: ");
    fgets(pass, sizeof(pass), stdin);
    pass[strcspn(pass, "\n")] = 0;

    if (strcmp(pass, clave) != 0) {
        printf("\n Contraseña incorrecta. Intruso detectado.\n");
        sleep(1);
        printf(" Sistema bloqueado. Salida de emergencia activada.\n");
        return 1;
    }
    acceso_concedido();
    f = fopen("/opt/secretos_del_banco.txt", "r");
    if (!f) {
        perror("Error al acceder al archivo");
        return 1;
    }
    char linea[256];
    while (fgets(linea, sizeof(linea), f)) {
        printf("%s", linea);
        usleep(300000); // efecto de máquina de escribir
    }
    fclose(f);
    printf("\n Fin del archivo. Desconectando...\n");
    return 0;
}
```

Figura 101: Código caja fuerte

Los permisos de este ejecutable los hemos configurado mediante estos dos comandos:

sudo chown root:root ./caja_fuerte

sudo chmod 4755 ./caja_fuerte

Tal y como podemos ver en el código, al ejecutarlo y lograr introducir la contraseña correcta, se muestra el mensaje de acceso concedido, junto al contenido del fichero secretos del banco creado antes y que es el objetivo de este hackatón.

Este ejercicio demuestra que la adaptación de nuestro entorno a nuevos hackatones es sencilla, flexible y reutilizable. La infraestructura creada ofrece una base sólida para diseñar nuevos escenarios personalizados, permitiendo modificar objetivos y configuraciones sin necesidad de reconstruir el entorno desde cero.

5.3 Reproducibilidad:

Una vez configurado el entorno y listo en su versión final, el paso a seguir es conseguir exportar esta versión de cara a proporcionársela a todos los participantes del evento. La forma más práctica y controlada de lograr esto es utilizando una máquina virtual preconfigurada con todo el entorno ya instalado. A continuación, se detallan las principales formas de reproducir el entorno:

Una vez configurado completamente el entorno en VirtualBox (con Ubuntu, las herramientas instaladas, reglas aplicadas y scripts configurados), puedes exportar toda la máquina virtual como un archivo .ova. Este archivo puede ser importado en cualquier otro ordenador que tenga VirtualBox instalado.

Para exportar el entorno, lo primero que habría que hacer es abrir VirtualBox y seleccionar la máquina virtual configurada. Después, se accede a Archivo > Exportar servicio virtualizado, en donde ponemos el nombre y la ubicación del archivo .ova.

Guardamos el archivo y ya lo podremos transferir a otros ordenadores (por USB, red local o nube). En los otros equipos, bastará con importar el .ova desde Archivo > Importar servicio virtualizado, iniciar la máquina virtual, y el entorno estará listo para funcionar.

Este método garantiza que el sistema operativo esté correctamente configurado, que todas las herramientas y scripts estén instalados, que las reglas de auditd y los servicios de systemd estén ya definidos y que el usuario sólo necesite introducir las credenciales y lanzar la herramienta.

Otra opción viable, aunque es más técnica, es crear un script de instalación automatizado (por ejemplo, un .sh) que configure el entorno desde cero sobre cualquier instalación de Ubuntu 24.04 limpia.

Este script podría ejecutar de manera automática la instalación de dependencias (apt install, pip install, etc.), la clonación de los repositorios desde GitHub y la creación del entorno virtual y activación, incluyendo las herramientas y su configuración.

Aunque más flexible, esta opción requiere que los equipos ya tengan Ubuntu instalado y acceso a internet. Es útil si no quieres depender de imágenes grandes como el .ova.

Capítulo 6 - Hackatón

De cara a poner en práctica nuestro entorno, hemos llevado a cabo una propuesta de hackatón a solucionar. Este hackatón lo hemos puesto a prueba con varios compañeros de la facultad con interés en la ciberseguridad, tal y como veremos más adelante.

A lo largo de este capítulo, explicaremos cómo hemos diseñado este hackatón y su ejecución, e incluiremos los resultados obtenidos tras la prueba con una serie de participantes que lo han realizado. Por lo tanto, este capítulo estará compuesto por el diseño del hackatón, la ejecución del mismo, las métricas de puntuación y las diferentes pruebas.

6.1 Diseño del hackatón

Un hackatón es un evento donde se trabaja en la resolución de un problema planteado en un tiempo limitado. En nuestro caso, el tema se centra en la seguridad informática y análisis de sistemas, más concretamente, el desafío es identificar y explotar una vulnerabilidad que permita a los participantes escalar privilegios en un sistema Linux, mientras nuestras herramientas detectan acciones sospechosas.

La idea con la que inició el diseño consistía en lo siguiente: Primero, se explica a todos los participantes el tema del evento. Luego, cada participante, en su ordenador, instala una máquina virtual preconfigurada; su tarea es analizar esta máquina, investigar las posibles fallas de seguridad y tratar de explotarlas para obtener acceso elevado en el sistema. Una vez que se ha conseguido, se redacta el proceso seguido y se avisa a un responsable para realizar el informe final. Finalmente, se hace una reunión de cierre frente a los participantes, en la que se valoran los resultados obtenidos y se daría por concluido el evento.

Una vez indicada la idea principal, toca explicar cuál es el criterio de elección de los participantes del hackatón. El perfil que buscamos es el de una persona con conocimientos básicos de ciberseguridad y con interés en mejorar sus habilidades en este ámbito. Respecto a su elección, principalmente estamos interesados en aceptar a

estudiantes que estén cursando la carrera de ingeniería informática o relacionadas, debido a que pensamos que la mayoría de los concursantes interesados en participar cumplen con estas características. Sin embargo, aún estamos valorando si esta característica es necesaria o no. Finalmente, este entorno está pensado para que sea aplicado a los laboratorios de la facultad de informática de la Universidad Complutense de Madrid, por lo tanto, otro de los requisitos es que los participantes han de residir en Madrid, o se han de desplazar en las fechas establecidas, ya que el evento no está pensado de momento en aceptar concursantes de manera telemática.

Finalmente, respecto a la infografía del evento, se ha realizado en su totalidad con Microsoft Word para redactar la información y las bases, y PowerPoint para facilitar la exposición del evento de cara a los participantes. Estos documentos, incluidos en los anexos, los hemos considerado necesarios para poder llevar a cabo este hackatón. La presentación de PowerPoint está planteada para explicar a los participantes de manera inicial en qué consiste el hackatón, los objetivos a cumplir y el tema específico. Esto ayuda a que todos se encuentren en el mismo punto desde el inicio del evento. Seguidamente, el Word es un documento detallado que explica las normas, las bases y el formato del hackatón, es decir, el cronograma del evento, las fases del hackatón, los recursos disponibles e información de contacto.

6.2 Ejecución del hackatón

Para ejecutar el hackatón, es necesario tener el entorno instalado en el ordenador donde se pretende realizar el hackatón, verificar que en él se encuentran los cambios descritos en el capítulo anterior y que las herramientas desarrolladas funcionan correctamente. Finalmente, habría que ejecutar la maquina Audit-Tool para obtener una primera imagen del sistema. Esto es necesario replicarlo para cada estación en la que trabajará cada participante.

Una vez iniciado el evento, se expone a todos los participantes, con ayuda de una presentación, una explicación del entorno que vamos a utilizar, los objetivos del evento y las normas a seguir. Además, consideramos que sería conveniente realizar una breve

formación práctica en la que expliquemos cómo no romper el entorno (que no tocar) y unas pautas sobre la ética de lo que se va a realizar, por ejemplo.

A continuación, los participantes procederían a trabajar sobre los equipos asignados para intentar cumplir el cometido.

En primer lugar, habrá una explicación teórica de conceptos básicos que son necesarios saber (permisos, usuarios, grupos, comandos básicos, archivo /etc/shadow y similares, bit SUID...). Posteriormente se dejará a los participantes unos minutos para investigar por su cuenta, en los cuales si tocan algo que no es debido como por ejemplo /etc/shadow, nos saltará una alerta a nuestro correo gracias a una de nuestras herramientas. El sistema está configurado como si fuese el ordenador de una sucursal bancaria, con numerosas carpetas y archivos relacionados. A continuación, siguiendo los pasos dados, el participante deberá ejecutar el comando cat en el archivo tarjetas_bancarias.txt, en el que se encuentra escondida la contraseña 'h4ck3r2024'. En el momento en el que acceda a ese archivo nos saltará una alerta indicando que ha picado en el honeypot.

Para buscar el archivo secreto con el bit SUID activado, lo primero que debe hacer es ejecutar el siguiente comando: find / -type f -perm -4000 2>/dev/null

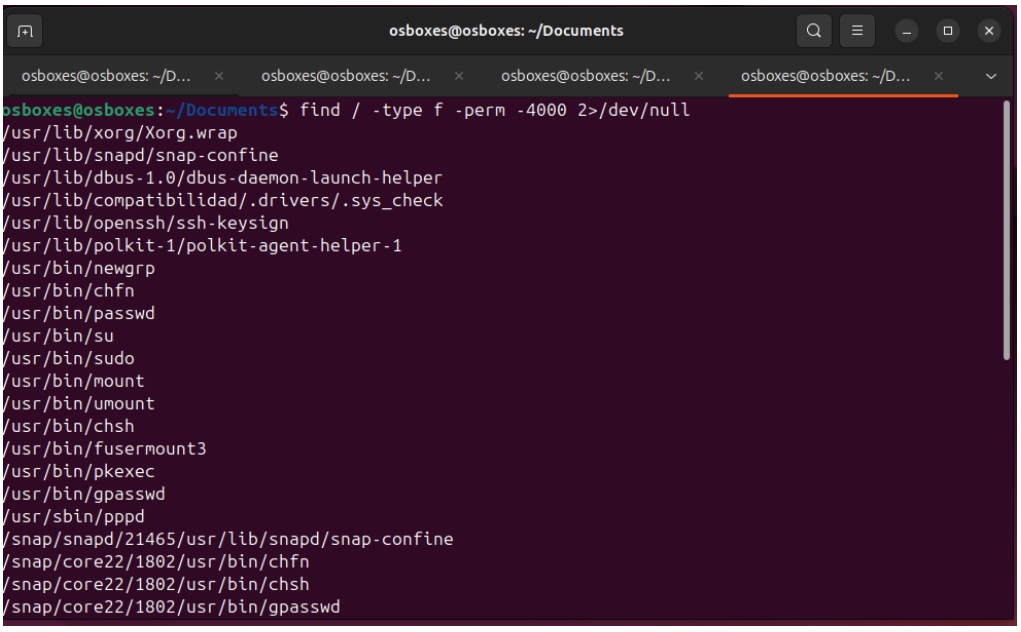

Figura 102: Ejecución Comando Búsqueda

Este comando le devolverá todos los directorios del sistema con el bit SUID activado.

El participante tendrá que darse cuenta que el correcto es /usr/lib/compatibilidad/.drivers/.sys_check, viendo que es el único escrito en español y con archivos ocultos (el . que se encuentra delante de drivers y sys_check indica que se encuentra oculto en el sistema) o probando por iniciativa propia.

Este binario tiene que ser ejecutado introduciendo la contraseña encontrada en tarjetas_bancarias.txt y ya habrá superado el hackatón.

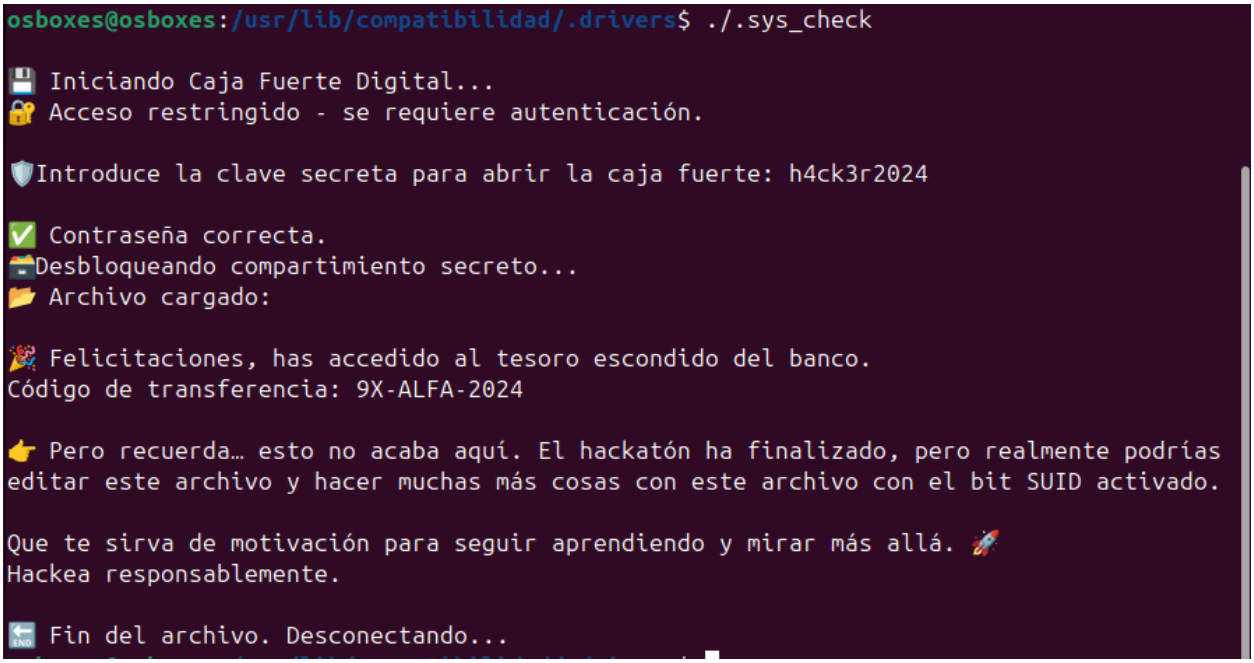

Figura 103: Final del hackatón

Pero como vemos en la Figura 101, el hackatón no termina ahí y para los más avanzados habilitamos un nivel superior. Consiste en modificar el .c oculto o crear uno nuevo con código malicioso para escalar privilegios o simplemente para ver el contenido de /etc/shadow; recompilar y ejecutar de nuevo .sys_check. Finalmente, habiendo conseguido los objetivos indicados al principio del evento, se podrían dar por concluidas las acciones del participante. Este deberá avisar a un responsable del evento para que verifique que se ha cumplido correctamente el cometido del hackatón y se procederá a recoger el estado final del sistema mediante la herramienta Audit-Tool. Una vez que todos los participantes hayan finalizado, se procedería a realizar una breve sesión de

cierre con los participantes dando feedback, hablando de la resolución del reto y presentando de manera conjunta unas conclusiones.

Tras esto, habría que analizar las diferencias entre el estado inicial y el estado final, para revisar si se hizo correctamente o si tocó algo que no debía. Se compararía con el resto de las soluciones de los participantes para valorar quien sería el ganador, usando diferentes métricas, como el tiempo de resolución. Finalmente, se valoraría el comportamiento del sistema y de las herramientas, para ver si el rendimiento, la estabilidad y la precisión fue correcta, de esta manera, se podría hacer un seguimiento del entorno.

6.2.1 Métricas de Puntuación

Para cuantificar el rendimiento de cada participante de forma objetiva y comparable, proponemos las siguientes métricas de scoring:

En primer lugar, se mide el tiempo de resolución, que corresponde al intervalo transcurrido desde el inicio de la sesión de hacking hasta que el participante introduce correctamente la contraseña del honeypot. Un tiempo más reducido indica rapidez en la identificación de la vulnerabilidad y en la ejecución de los comandos necesarios.

En segundo lugar, se contabiliza el número de alertas disparadas por la herramienta de monitorización. Cada intento de lectura del honeypot genera una alerta, de modo que un menor número de alertas refleja un mayor sigilo y precisión en la estrategia de ataque.

Estas cinco métricas, combinadas con un sistema de ponderaciones acordadas (por ejemplo, 30 % tiempo, 25 % alertas, 25 % informe, 10 % comandos efectivos y 10 % creatividad), permiten obtener una puntuación global que refleje tanto la rapidez como la calidad y profundidad de la actividad de cada concursante.

6.2.2 Cronograma y Roles

Hora	Actividad	Responsable (Rol)
09:00–09:15	Registro de participantes y comprobación de VMs	Facilitador
09:15–09:30	Bienvenida y briefing: objetivos, reglas y ética	Facilitador
09:30–09:45	Formación exprés: uso de AuditTool, MonitoringTool y DefenseTool	Facilitador & Blue Team
09:45–10:00	Ejecución inicial de Audit-Tool (estado base)	Participantes; verifica Blue Team
10:00–12:00	Sesión de hacking	Participantes
	Monitorización continua (MonitoringTool y DefenseTool)	Blue Team
	Soporte y FAQ en canal Discord/WhatsApp	Facilitador
12:00–12:15	Pausa breve	—
12:15–12:30	Ejecución final de Audit-Tool (estado post-hack)	Participantes; supervisa Blue Team
12:30–12:45	Revisión de flags y comprobación de logs	Juez(s)
12:45–13:15	Debriefing y cierre: presentación de resultados y aprendizajes	Facilitador, Blue Team, Juez(s) y Participantes

Tabla 2

A continuación, se describen las responsabilidades de cada rol definido en el cronograma:

El **facilitador** asume la organización y coordinación global del evento: se encarga de dar la bienvenida a los participantes, registrar su asistencia y verificar que las máquinas virtuales estén correctamente desplegadas. Durante el briefing inicial, explica los objetivos del hackatón, detalla las reglas de participación, repasa las normas éticas y ofrece una introducción al uso de las herramientas Audit-Tool y Monitoring_Tool. A lo largo de la jornada, mantiene abierto un canal de comunicación (Discord/WhatsApp) para resolver cualquier incidencia técnica o consulta, actuando además como plan alternativo ante problemas imprevistos.

El equipo de "defensa" o **Blue Team** vela por la seguridad del entorno y la integridad de los datos generados. En tiempo real, monitoriza el tráfico del *honeypot* y analiza los eventos que dispara la Monitoring_Tool, alertando inmediatamente de cualquier acceso no autorizado. Asimismo, supervisa la ejecución de Audit-Tool tanto al inicio como al cierre del hackatón para garantizar que las capturas de estado del sistema sean fiables. Durante la fase de formación exprés, presta apoyo al facilitador y ayuda a los participantes a resolver dudas de configuración.

Quienes actúan como **juez** o panel de jueces revisan las evidencias aportadas por cada concursante: las flags encontradas, los registros de log y los informes generados por Audit-Tool. Corroboran la validez de cada flag según las métricas establecidas (tiempo de resolución, número de alertas detonadas y calidad del informe) y asignan las puntuaciones correspondientes. Finalmente, elaboran un resumen de resultados para el *debriefing*, destacando los aciertos de los participantes, los errores comunes y las principales lecciones aprendidas.

Cada **participante** dispone de una máquina virtual preconfigurada en la que ejecuta las técnicas de explotación necesarias para descubrir la contraseña del *honeypot* y aprovechar el bit SUID. Antes de iniciar el hacking, lanza Audit-Tool para capturar el estado base del sistema, y repite la misma operación al finalizar para documentar las modificaciones realizadas. Durante todo el proceso, registran sus pasos en un informe breve —incluyendo capturas de pantalla y fragmentos de log relevantes— que entregan al juez al concluir la prueba.

6.3 Pruebas del Hackatón

En este apartado describiremos la realización del hackatón con los participantes que han realizado la prueba. Hablaremos sobre las cosas que han sido necesarias para celebrarlo en la realidad, el espacio donde lo hicimos, las dificultades que tuvieron, los fallos que encontramos y la satisfacción de los participantes.

Se han llevado a cabo dos pruebas, una con alguien con nivel básico (estudiante informática sin grandes conocimientos en ciberseguridad), y otra con un estudiante avanzado con grandes conocimientos en ciberseguridad. Los resultados que obtuvimos en ambos entornos de prueba nos sirvieron para acabar de perfeccionar y pulir algunos detalles que no estaban configurados al completo, destacando la prueba con el estudiante avanzado.

En primer lugar, se celebró el hackatón con el estudiante de nivel básico. Después de la explicación, tuvo unos 10 minutos para que se hiciese al entorno e investigase por su cuenta lo que quisiese. Al explicarle en el apartado teórico que existía un archivo con los usuarios y sus contraseñas hasheadas (etc/shadow) fue a lo primero a lo que intento acceder, saltando así la alerta en nuestro correo generada por Defense_Tool. A continuación, se puso a investigar en la carpeta *Documents*, donde existen numerosos archivos relacionados con contabilidad y actividades bancarias. Al abrir uno de estos documentos saltaría la siguiente alerta generada por monitoring_tool. Realmente el archivo es un cebo, denominado *honeypot*, que está presente en tarjetas_bancarias.txt. En este documento encuentra la contraseña para proseguir con el hackatón. Finalmente, siguiendo con las indicaciones, buscó todos los archivos con el bit SUID activado y le llamo la atención la única carpeta con el nombre en español. Accedió a ella, pero sin éxito no encontró nada debido a que los archivos están ocultos. Pasados unos minutos se percató de este detalle y ejecutó el binario oculto. Al ejecutar ese archivo oculto le pidió la contraseña que aparecía en tarjetas_bancarias.txt. Acabando con el hackatón y sin proseguir con la fase avanzada debido a la extensa duración de la prueba para el participante.

El segundo hackatón se celebró con el estudiante con conocimientos avanzados en ciberseguridad. Después de la explicación fue mucho más allá, el ansia le pudo y se

puso a analizar todo al completo. Incluso consiguiendo acceder a un archivo que no debería y que corregiríamos posteriormente. Realizó el resto de la prueba con normalidad, exceptuando que al ejecutar el binario final no se dio por vencido y quiso seguir la pista oculta que dejamos al finalizar; esta menciona que el archivo con el bit SUID activado tiene algunos permisos mal configurados y puede ir más allá para escalar privilegios o hacer lo que se le ocurra. Para este paso final necesitó mucho más tiempo, pero lo logró finalmente, lo que hizo fue recompilar en .sys_check un nuevo archivo llamado pruebas.c con código que le permitía acceder a /etc/shadow sin tener permisos. Esto sucede debido a que .sys_check tiene el bit SUID activado y puede ejecutar el archivo con privilegios root. El estudiante quedó muy satisfecho con el hackatón y lo aprendido en él, incluso más que el primer participante.

Ambos hackatones se celebraron sin ninguna incidencia y con gran satisfacción por ambas partes en todos los aspectos. Al finalizar, les hicimos una serie de preguntas y si nos podían dar alguna sugerencia. El primer estudiante no aportó ninguna sugerencia; sin embargo, el segundo sí lo hizo, sugiriendo más archivos de despiste que no sirviesen para nada, pero que hiciesen pensar al participante. Las sugerencias fueron aceptadas e implementadas en el entorno final, además de las correcciones correspondientes a la configuración.

Capítulo 7 - Conclusiones, trabajo futuro y aportaciones personales.

En primer lugar, antes de empezar a tratar el desarrollo técnico, el proyecto se inició con una planificación detallando los roles y responsabilidades principales que llevarían a cabo cada uno de los integrantes. Esta organización nos permitió que todas las partes avanzasen en paralelo y de manera coordinada, con un seguimiento facilitado por las herramientas utilizadas: GitHub, Google Drive y Discord. Desde un inicio, la prioridad fue la búsqueda de información relevante, sobre todo en documentación oficial, diferentes hackatones y otros proyectos similares. Este proceso nos permitió fundamentar las decisiones técnicas y asegurar el buen enfoque en el diseño de nuestras herramientas y entorno.

En cuanto al apartado técnico y propio del proyecto, cabe destacar la importancia de la creación de la herramienta Audit-Tool, la cual es clave en la detección de cambios en el sistema entre el inicio y el final del hackatón. Esta herramienta genera una "imagen" inicial y final de las configuraciones del entorno y, para desarrollarla, tuvimos que comprender perfectamente todos los elementos del sistema operativo para su correcto funcionamiento. Esta herramienta nos sirve como una línea base de seguridad, permitiendo identificar vulnerabilidades y configuraciones erróneas o débiles del sistema. Para el desarrollo de esta herramienta fue clave la buena organización del código en módulos, lo que permite añadir más información acerca del sistema operativo de manera sencilla.

En segundo lugar, después de informarnos sobre las mejores técnicas de detección en un sistema, decidimos desarrollar un honeypot. Esto nos sirve para atraer a los participantes hacia un objetivo específico, demostrando así la efectividad de esta técnica de detección y respuesta ante incidentes. Esta técnica ayuda a reflexionar a las dos posiciones contrarias: tanto al atacante para que no confíe en todo lo que intenta abrir y al defensor en implementar una buena detección y respuesta ante ataques. Para implementar el honeypot, fue muy importante comprender muy bien cómo se creaba un servicio permanente en el sistema operativo y que este mismo

estuviese oculto con sus respectivos permisos configurados. A continuación, gracias al buen funcionamiento de Audit-Tool, se puede localizar el binario oculto con el bit SUID activado, demostrándonos que una simple mala configuración puede comprometer el sistema al completo.

La última herramienta desarrollada fue defense_tool, la cual nos permite detectar en tiempo real el intento de acceso a cualquier archivo sensible, obteniendo así una gran robustez en nuestra detección y respuesta. Esta herramienta es importante tanto para proteger el sistema como para el correcto desarrollo de hackatones, ya que permite a los organizadores bloquear los archivos sensibles del sistema a los que no quieren que los participantes accedan y alertar sobre ello. Para su desarrollo, nos basamos en la implementación de monitoring_tool, ya que la lógica de funcionamiento es muy similar. También actúa como servicio, por lo que se encuentra presente en el sistema operativo una vez este se inicie. Todo esto fue desarrollado en el mismo entorno que el presente en los ordenadores de los laboratorios de la Facultad de Informática, por tanto, es perfectamente replicable, formando parte de nuestro trabajo futuro.

Todo esto refleja la importancia de la implementación de monitoreos continuos y mejora en la concienciación en ciberseguridad para adoptar mejores prácticas para proteger sistemas. Ya que, si una simple configuración errónea compromete el sistema al completo, tanto la solución cómo los problemas y errores se encuentran en las personas. Por ello, concluimos que es fundamental que todas las personas involucradas en el desarrollo, administración o uso de un sistema informático sean conscientes de que incluso un pequeño fallo se puede convertir en un error grave o en una brecha de seguridad crítica. La ciberseguridad no depende solamente de las herramientas tecnológicas, sino también del conocimiento y las buenas prácticas de toda persona que utilice un sistema informático.

Por último, tenemos que destacar la importancia de haber cursado las asignaturas de Redes y Seguridad I y II, Redes, Sistemas Operativos, Ampliación de Sistemas Operativos y Redes, Auditoría Informática I y II y Evaluación de Configuraciones, que nos proporcionaron una base sólida para dar los primeros pasos del proyecto y nos permitieron profundizar más en diferentes aspectos, teniendo una visión más amplia de

los problemas y soluciones que se presentan hoy en día en el mundo de la ciberseguridad.

7.1 Trabajo Futuro

La creación de este entorno no es más que el inicio de lo que podría ser un proyecto aún mayor. Todo el trabajo y entorno está enfocado en su implementación futura en los laboratorios de la Facultad de Informática, por ello nuestros siguientes pasos irían en esa dirección, trasladando todas las herramientas y configuraciones; esta réplica del trabajo realizado sería muy sencilla debido a que el sistema operativo empleado es el mismo. Tras haber validado nuestra propuesta de hackatón, tal y como vimos en el anterior capítulo, estamos motivados para organizar un evento oficial y presencial, con un diseño de mayor complejidad y un periodo de planificación más extenso. Sin embargo, entre los próximos desafíos se encuentran la obtención del visto bueno de la Facultad, la captación de patrocinadores y la coordinación de recursos a mayor escala.

Aunque hemos concebido esta iniciativa inicialmente para un piloto a pequeña escala, su potencial trasciende el ámbito académico. Imaginamos la celebración de hackatones corporativos que reúnan a cientos de participantes, convirtiéndolo incluso en una oportunidad de negocio. Los tres integrantes del grupo hemos cursado la asignatura de Creación de Empresas con el profesor David Pascual, y desde entonces barajamos convertir esta experiencia en una iniciativa emprendedora. Por ello, durante las vacaciones de verano evaluaremos si nuestra ilusión se traduce en un modelo de negocio viable o si, por el contrario, ha sido simplemente la satisfacción de culminar un gran reto.

7.2 Aportaciones Personales

Federico López

Mi contribución en el Trabajo de Fin de Grado ha sido completa en todas las fases de su desarrollo, tanto en el diseño general como en la implementación de herramientas y su respectiva documentación. A continuación, se detallan las tareas que he realizado durante la ejecución del TFG.

Desarrollo completo de la herramienta *defense_tool*

He sido el responsable directo del diseño, implementación y pruebas de la tercera herramienta, llamada defense_tool, cuyo objetivo es detectar accesos no autorizados a archivos sensibles durante la ejecución del hackatón en tiempo real. Para su implementación, me basé en la de monitoring_tool, y en una primera instancia mi objetivo iba a ser modificar el código de esa herramienta añadiendo la nueva funcionalidad. Después de intentarlo con distintos enfoques, decidí crear una herramienta totalmente nueva y que funcionase de manera independiente a monitoring_tool. De esta manera, también facilitaría añadir más archivos sensibles que se quieran proteger dependiendo de cada hackatón. Por lo tanto, en su versión final, la herramienta se compone de un script principal que ejecuta un monitoreo cíclico del sistema y, en caso de detectar un intento de lectura no autorizado, realiza varias acciones: registra el evento en un log, envía una alerta por correo, bloquea el archivo y clasifica el intento como exitoso o fallido. Además, guarda un historial de eventos procesados para evitar repeticiones.

Para garantizar su ejecución automática y continua, también desarrollé el archivo defense.service, que permite ejecutar la herramienta como un servicio del sistema mediante systemd, iniciándose al arrancar el sistema y funcionando en segundo plano como un demonio. Todo esto me dio varios problemas de funcionamiento, hasta que fui capaz de solucionar todos los errores y probar su correcto funconamiento en el archivo /etc/shadow.

Aunque fue pensada para el entorno del hackatón, defense_tool es una solución funcional y escalable que puede aplicarse en entornos reales, permitiendo añadir más archivos a monitorizar y reforzando la seguridad del sistema.

Diseño, desarrollo y ejecución del hackatón

Otra de mis responsabilidades fue, diseñar y ejecutar un hackatón para poner a prueba las herramientas implementadas. Desde un primer momento, tuvimos claro que iba a estar relacionado con un ataque al bit SUID, pero no sabíamos cómo llevarlo a cabo. En una primera instancia, cree un archivo .c con el bit SUID activado y código ejecutable

en su interior que permitía acceder a archivos con permisos de root siendo un usuario normal. A partir de esto, llegamos a la conclusión de que debíamos esconder de alguna forma el archivo .c y que el participante del hackatón tuviese que probar diferentes cosas hasta encontrarlo. Para ello, creamos un archivo .txt muy a la vista llamado tarjetas_bancarias, el cual haría que, una vez el participante accediese a él, saltaría la alerta de monitoring_tool. En tarjetas_bancarias.txt decidimos "esconder" una contraseña sin explicar para qué iba a servirle al participante que abriese el archivo. Después, escondimos el .c por el sistema operativo, cambiándole el nombre para no levantar sospechas del participante y al ejecutarlo pedía la contraseña que habíamos obtenido antes en tarjetas_bancarias.txt. Para encontrar el archivo oculto con el bit SUID activado, el participante debía ejecutar un comando que se los mostrarse, y darse cuenta de cuál era el archivo "raro", ya que la ruta indicaba que había cosas ocultas. Una vez encontrado e introducido la contraseña, el hackatón habría finalizado correctamente. Por último, decidimos introducir una pista oculta al finalizar el hackatón, mencionando que el archivo con el bit SUID activado tiene algunos permisos mal configurados y el usuario puede intentar escalar privilegios u otra acción aprovechándose de ello. De esta forma, si el participante es capaz de recompilar código en el archivo con el bit SUID activado, podrá realizar acciones de root sin serlo.

Revisión, análisis y documentación del código de las tres herramientas

Además del desarrollo de la herramienta defense_tool, fui en el encargado de la revisión exhaustiva, pruebas y documentación de las otras dos herramientas (Audit-Tool y monitoring_tool). Gracias a ello, fui capaz de comprender con total claridad el funcionamiento y lógica de cada una de ellas. Para realizar la documentación técnica, me centré en la explicación de la estructura modular de cada una, para qué se utilizan y la explicación detallada de todo el código de cada una de ellas.

La mayor complicación fue entender a la perfección el lenguaje Python, ya que no lo había visto a lo largo de la carrera y tuve que investigar acerca de él. Me facilitó mucho las cosas el tema de que sea tan parecido a otros lenguajes que había visto con anterioridad. En esta documentación, está explicada cada función paso a paso, indicando para qué se usa y por qué es útil.

El documento más extenso es el de Audit-Tool, ocupando 83 páginas. Esto es debido a que está formada por muchos módulos extensos implementados mediante diferentes funciones. La documentación de monitoring_tool tiene una extensión de 9 páginas y la de defense_tool son 13 páginas.

Organización de los repositorios en GitHub y escritura de los README de las herramientas

Con el objetivo de profesionalizar y estructurar de una forma clara el proyecto, me encargué de organizar los repositorios de las tres herramientas en GitHub, así como de la redacción de sus respectivos README. Para realizarlo, me basé en los repositorios de otras herramientas de ciberseguridad, cómo estaban estructuradas y qué era lo importante para que quedase lo más profesional posible. Para ello decidí redactar cada archivo README.md en inglés, debido a que es el lenguaje más común en el mundo de la informática. En cada archivo, incluí una explicación general de la respectiva herramienta, así como la estructura de sus diferentes módulos. También decidí redactar paso a paso cómo se instalaba cada una y su forma de usarla. Para facilitar y amenizar la lectura del archivo, hice uso de diferentes emoticonos representativos. Por lo tanto, el objetivo de estos README es facilitar la comprensión y utilización de la herramienta para terceros, sin tener que revisar en profundidad el código fuente.

Redacción de contenidos de la memoria

Finalmente, me encargué de la redacción de varias partes de la memoria. En una primera instancia, me dediqué a buscar información acerca del lenguaje Python, ya que era el que íbamos a utilizar en la implementación de nuestras herramientas y es el más común en el mundo de la ciberseguridad. Para ello, me centré en explicar las características generales del lenguaje y el por qué es útil en el mundo de la ciberseguridad. También busqué información acerca de diferentes bibliotecas y para que se usa cada una de ellas. Por último, destaqué la importancia de Python en la automatización de tareas en el ámbito de la ciberseguridad y cómo se implementa un caso de uso para preparar un hackatón. Redacté las bases de los capítulos de desarrollo de las herramientas, explicando cada una de ellas de una manera más informal para su futura correcta redacción. También me encargué de explicar detalladamente la instalación y uso de cada una de ellas, así como los respectivos cambios necesarios en el sistema operativo para su uso. Fui el encargado de realizar las diferentes pruebas para

la ejecución del hackatón y su correspondiente documentación con el paso a paso para resolverlo y las modificaciones del sistema operativo para dejarlo preparado para su ejecución. Por último, revisé y ayudé a redactar las conclusiones acerca del proyecto completo.

Manuel Louro

Responsable de la fase de investigación previa, desarrollo al completo de las herramientas audit-Tool y monitoring_tool, diseño, desarrollo y ejecución del hackatón, escritura de capítulos de la memoria y revisión total del contenido de la misma.

Primeros pasos del proyecto

La primera de mis contribuciones en este proyecto ha sido la investigación previa necesaria para ubicar los objetivos y el marco de trabajo, que sería trasladado a mis compañeros para seguir una hoja de ruta adecuada. Después de esta fase de investigación y la recopilación de toda la información y recursos, destacando la máquina virtual usada; estuve varios días haciéndome al entorno e investigando por mi propia cuenta, para poder encontrar elementos de mejora e ideas a implementar en el proyecto sin influencia de una investigación profunda en este aspecto. Posteriormente, ya con la idea proyectada al completo, procedo a desarrollar la primera herramienta, audit_tool.

Desarrollo completo de la herramienta audit_tool

Responsable del diseño, desarrollo e implementación completa de la herramienta audit_tool, cuyo objetivo es realizar una auditoría integral de seguridad en sistemas operativos Linux, en este caso, especialmente Ubuntu. Esta herramienta ha evolucionado significativamente a lo largo del proyecto, atravesando diversas fases de desarrollo que fueron clave para su maduración técnica y modularidad.

En primer lugar, desarrollé una versión de prueba con un único archivo con toda la lógica, sin ningún tipo de organización previa. Esto suponía que el proyecto fuese muy poco escalable y por cada error se comprometía el código al completo, por ello para la siguiente versión decidí aplicar la modularidad y gestionar cada módulo

independientemente. Esta versión fue la v1.0, separada por módulos con ejecución independiente de las diferentes secciones, siendo implementados los 10 primeros (sys_info, kernel, mem_process, users_groups_auth, debian_tests, boot_services, network, file_permissions, home_directories, storage_device). Este nuevo enfoque fue el punto clave para un buen desarrollo de la herramienta, permitiendo la adición de nuevos módulos y la prueba de los mismos con facilidad.

En este punto considero que termino esta versión debido a que constituye la base principal de la herramienta, enseñándosela a mis compañeros para que realizasen las pruebas pertinentes. La siguiente versión, v1.1 no difiere demasiado de la anterior, debido a que trabajo sobre la base anterior con el mismo enfoque modular y añado 10 nuevos módulos: logs, app_security, advanced_network_security, service_accounts, containers_security, updates, security_policies, sudo, storage_services, malware_protection, backup. Estos nuevos módulos proporcionan una cobertura ampliada del sistema, obteniendo también mayor granularidad.

La última versión es la 2.0, la cual supone una reorganización de los módulos en 9 categorías temáticas, teniendo así una organización lógica con respecto a los resultados esperados. Además, en esta fase, se mejora la salida de los datos por consola y la organización del archivo JSON final.

Desarrollo completo de la herramienta monitoring_tool

Responsable del diseño, desarrollo e implementación completa de la herramienta monitoring_tool. Esta herramienta de monitorización basada en técnicas de honeypot, tiene como objetivo detectar accesos no autorizados a archivos sensibles en sistemas Linux en tiempo real.

La herramienta está basada en el concepto de honeypot de archivo, una técnica que consiste en crear un archivo trampa, el cual su apertura no debe producirse en condiciones normales. Cualquier intento de acceso se considera una señal clara de comportamiento anómalo. Este enfoque, muy usado en entornos de Active Directory, permite detectar intrusiones de forma directa y eficaz. En primer lugar, creé el archivo señuelo (tarjetas_bancarias.txt) ubicado en una ruta muy reconocible del sistema para

que cualquier usuario pudiese acceder a él. Al archivo le otorgo permisos de lectura para que cualquier usuario pueda entrar y caer en el honeypot, alertándonos al instante.

A continuación, desarrollé la herramienta monitoring_tool, que se ubica en una ruta protegida para que se ejecute como servicio automáticamente. Para el desarrollo de la herramienta tuve en cuenta que esta serie de objetivos estuviesen cubiertos: lectura de eventos auditados, extracción de información importante como usuario e ip, bloqueo del archivo y envío de la alerta por correo electrónico al instante. Para que todo esto funcionase tuve que configurar el sistema de auditoría auditd, del cual haría la lectura de los logs y por último, utilizando systemd, convertiría el script finalmente en un servicio que se ejecutase en todo momento desde el arranque del sistema.

Diseño, desarrollo y ejecución del hackatón

Otra de mis responsabilidades, fue diseñar y ejecutar el hackatón al completo para poner a prueba las herramientas implementadas. Esta tarea la realizamos ambos conjuntamente desde el primer momento que propuse la idea inicial que iríamos modelando entre ambos con el desarrollo del mismo. La explicación del proceso total está reflejada al completo previamente en las aportaciones de mi compañero.

Revisión y finetunning de la tercera herramienta, defense_tool

Esta herramienta fue diseñada al completo por mi compañero Federico, pero debido a algunos errores finales fui el encargado de subsanarlos conjuntamente con él.

Redacción de contenidos y revisión completa de la memoria

En cuanto a la memoria, estuve involucrado en todas sus fases y apartados, tanto en la redacción de varios de los capítulos, como en la revisión de la misma al completo en numerosas ocasiones.

Pablo Sevillano

Mi contribución en el Trabajo de Fin de Grado ha sido completa en todas las fases de su desarrollo, asumiendo un papel más centrado en la documentación y

acompañamiento durante las diferentes fases. A continuación, se detallan las tareas que he realizado durante la ejecución del TFG.

Organización y metodología de trabajo

Desde el inicio del proyecto asumí la responsabilidad de definir y estructurar la metodología de desarrollo, articulando un modelo de trabajo híbrido inspirado en prácticas ágiles. Para ello dediqué un tiempo a informarme sobre las metodologías de trabajo estudiadas a lo largo del curso, llegando a la conclusión de que la metodología de SCRUM (Puesta en práctica durante la asignatura de Desarrollo de Sistemas Interactivos), iba a ser la mejor se adaptara a nuestro estilo de trabajo. Para ello, diseñé un sistema de reuniones semanales ("sprints") y creé canales de comunicación en Discord y WhatsApp para coordinar tareas, resolver dudas y tomar decisiones rápidas. Además, configuré y gestioné Google Drive como repositorio central de la documentación. Gracias a este marco, todo el equipo pudo avanzar de forma paralela y sincronizada, con visibilidad completa de los hitos alcanzados y los próximos pasos a seguir.

Diseño y redacción de la memoria

Fui responsable de la redacción íntegra de la memoria final, que abarca desde la investigación de los requisitos de la memoria hasta la redacción desde la motivación y el estado del arte hasta las conclusiones y trabajo futuro. Tras recopilar la información inicial, estructuré el índice y definí el contenido de cada capítulo, asegurando coherencia entre secciones y transiciones claras. Redacté las explicaciones técnicas de cada herramienta, describiendo su diseño, el proceso del desarrollo, el funcionamiento y las pruebas principales realizadas, cuidando el estilo académico para que resultara legible y riguroso. Además, integré tablas y capturas de pantalla de manera alineada con las normas de formato, facilitando la lectura y la comprensión por parte del tribunal.

Recopilación y gestión de la información

Coordiné la investigación bibliográfica y documental que sustenta el TFG. Localicé y evalué fuentes oficiales (manuales de Ubuntu, documentación de Python, guías de herramientas de ciberseguridad) y organicé su cita en el formato estándar. Validé cada fuente para garantizar su actualidad y fiabilidad, evitando contenidos obsoletos o poco

relevantes. Esto fue una tarea que llevó bastante tiempo ya que, a lo largo de la investigación de este trabajo, se llegaron a registrar cientos de enlaces a fuentes bibliográficas que tuvieron que ser revisadas y filtradas, según la relevancia de las mismas. Mi labor de curación documental fue esencial para fundamentar las decisiones de diseño y otorgar solidez académica a toda la memoria.

Diseño y documentación del hackatón y anexos

Participé en el diseño del esquema completo del hackatón: definición de objetivos, dinámica, rutas de archivos y pistas ocultas. Limitándome principalmente a la toma de decisiones, acompañamiento y a la documentación necesaria para el hackatón propuesto (el panfleto informativo en Word, las bases legales con el marco jurídico aplicable y la presentación en PowerPoint para la sesión introductoria), además de redactar el capítulo prácticamente entero. Organicé todos los materiales complementarios y los agrupé en la sección de Anexos de la memoria, garantizando referencias cruzadas claras. Esta labor de documentación permitió al tribunal y a los futuros organizadores reproducir el evento con facilidad.

Revisión total del contenido y énfasis en el análisis

Realicé una revisión global de la memoria, corrigiendo estilo, ortografía, formato de tablas y numeración de figuras. Verifiqué la coherencia interna de cada sección y unifiqué terminología técnica para evitar ambigüedades.

Resolución de errores y recopilación de métricas

Para la redacción del capítulo 6 colaboré estrechamente en la identificación y corrección de incidencias, registrando meticulosamente cada fallo producido. Para cada problema, registré las distintas aproximaciones llevadas a cabo para cada herramienta, anotando el contexto de ejecución, comandos exactos utilizados y resultados obtenidos. Tras consolidar cada versión definitiva, contabilicé el número de commits asociados a cada corrección y medí el tiempo invertido en cada solución. Todas estas métricas (número de intentos, tiempo dedicado, volumen de código y nuevas dependencias) fueron incluidas dentro del informe, con comentarios explicativos sobre su significado. De este modo, proporcioné al tribunal una visión

detallada de la dificultad real del desarrollo y la evolución de las herramientas, fundamentando la complejidad del TFG en datos objetivos y trazables.

Introduction

Motivation

The laboratories of the School of Computer Science host a large number of applications and are used by numerous students, which can generate security vulnerabilities. Thus, it is essential for students to acquire practical knowledge on system protection and threat detection.

To address these challenges, controlled environments such as cybersecurity hackathons are proposed as an opportunity to experiment and learn, as well as a good strategy for students to put their knowledge into practice in a real and safe environment.

It is clear that, to ensure the proper use of the laboratories during these events, it is necessary to use tools that allow real-time monitoring and auditing of the equipment. Consequently, this Final Degree Project (hereinafter, FYP) arises from the motivation to equip the faculty's systems with an environment prepared to carry out a cybersecurity-oriented hackathon, allowing participants to face this type of scenario without compromising the integrity of the equipment or the faculty's network.

Goals

The objective of this Final Degree Project is to develop a solution based on the operating system used by the laboratory computers, including custom tools to audit the system and monitor sensitive access in real time. The aim is to promote active learning and demonstrate the importance of preparation, detection, and response to security incidents that may arise in the academic environment.

Specifically, the project aims to:

First, develop an auditing tool capable of analyzing the system state before and after the hackathon, allowing the identification of potential alterations and/or suspicious behavior. Additionally, it also aims to provide a comparison between the initial and final system states through reports generated by the aforementioned tool.

Second, create a real-time monitoring tool that can log and alert about access to sensitive files defined by the event organizer.

Third, implement a tool that defends the system against unauthorized access by participants, blocking their access and alerting the responsible party.

Fourth, configure and adapt the operating system of the faculty's computers to ensure the operation of a secure and controlled environment during the event.

Fifth, design an introductory training activity aimed at participants. This training would include basic cybersecurity concepts and the execution of a controlled attack on the SUID bit.

Finally, ensure the persistence and continuous operation of the monitoring tool as a system service.

Work plan

The development of our project followed a flexible and adaptive methodology, based on continuous collaboration and the assignment of roles within the team. Throughout the process, we implemented practices similar to agile methodologies, particularly an approach resembling Scrum [1], although without a rigid formal structure.

From the beginning of the project, our team was composed of three members with different responsibilities: One was responsible for the development and programming of the tools. Another was in charge of documentation, information gathering, and drafting the report. Finally, the third focused on managing the GitHub repository, the technical documentation of the tools, and the development of one of the tools.

Although at the start of the FYP the organization was not clearly defined, we progressively established a more efficient distribution of tasks, allowing each team member to specialize in a specific area.

Communication was key in the development of the project. To coordinate our work, we used a WhatsApp group, where daily progress and doubts were discussed, and we also

held weekly meetings on our Discord server, where we reviewed the progress and set new tasks.

These meetings functioned as weekly sprints, during which we analyzed the project's status and set the next objectives to complete. In this way, without following a formal framework, we implemented an iterative and adaptable work dynamic.

For project management and development, we used various tools that facilitated joint progress through telematic and synchronous collaboration. Regarding documentation, we used Google Drive as a storage and sharing platform for documents and resources used in drafting the report, which was created in Microsoft Word, utilizing the document sharing feature.

As for the tools' code, we used Visual Studio Code for development, and for publishing to our GitHub, where we stored our progress by versions and the tools themselves, we used GitHub Desktop. This application allowed us to publish the code directly without using commands, in a very simple way. Finally, we used VirtualBox to work with the specified Ubuntu system and to implement the environment described in this report. This application allowed us not only to run an operating system without installing it on our laptops but also to export this environment for use in future hackathons.

To achieve proper organization and an efficient distribution of the work plan, this project was structured into the following phases:

1. Preliminary Research

We began with a documentation and research phase to understand the context of cybersecurity hackathons, the concept of cybersecurity tools, the functioning of the operating system we used, and the programming language employed. This phase allowed us to clearly define the technical and functional objectives of the project.

2. Tool Design and Development

Based on the requirements established in the next section, three key solutions were designed and programmed:

- A system auditing tool to collect and analyze configurations and critical events.
- A monitoring tool to supervise the status of services and resources in real time.
- A defense tool capable of applying automatic countermeasures upon detection of anomalous activities.

3. Testing and Environment Validation

Tests were carried out in controlled environments to verify the correct functioning of each tool. These tools were then installed in a single environment, thus creating the base for holding cybersecurity hackathons.

4. Event Design and Execution

Based on the base environment, a test scenario was created as a first hackathon proposal, which allowed deployment on other machines. A pilot was then run on a team member's computer. The auditing tools were launched at the beginning and end of the event, while the monitoring and defense solutions operated in real time to validate their proper behavior.

5. Final Documentation: Report

To create this report, all documents generated during the research phase were collected, and the development process as well as the functioning of each tool were described in detail. Next, the steps followed to create the environment and conduct the hackathon test were documented. Finally, the results obtained and the conclusions derived from the project were presented.

To visually represent these phases and the time invested, we included a Gantt chart (Figura 1: Diagrama de Gantt) showing the temporal planning of each task throughout the project.

In addition to this diagram, we wanted to emphasize how the development progress unfolded, focusing more specifically on the design progress of the tools. To achieve this, we have chosen to include a second diagram below to visually represent this development (Figura 2: Diagrama de Gantt - Desarrollo herramientas).

As can be seen, the development phases overlapped with one another, as we were concerned about not completing all the proposed tools on time due to the complications that arose throughout the process.

Conclusions and future work

First of all, before beginning the technical development, the project started with a planning phase detailing the main roles and responsibilities that each member would carry out. This organization allowed all parts to progress in parallel and in a coordinated manner, with monitoring facilitated by the tools used: GitHub, Google Drive, and Discord. From the beginning, the priority was the search for relevant information, especially from official documentation, various hackathons, and other similar projects. This process allowed us to base technical decisions and ensure the proper focus in the design of our tools and environment.

Regarding the technical section and the core of the project, it is worth highlighting the importance of creating the Audit-Tool, which is key in detecting changes in the system between the beginning and the end of the hackathon. This tool generates an initial and final "snapshot" of the environment configurations, and to develop it, we had to fully understand all elements of the operating system to ensure correct functionality. This tool serves as a security baseline, allowing the identification of vulnerabilities and misconfigured or weak system settings. For its development, the good organization of the code in modules was essential, which makes it easy to add more information about the operating system.

Secondly, after researching the best detection techniques on a system, we decided to develop a honeypot. This helps attract participants toward a specific target, thus demonstrating the effectiveness of this detection and incident response technique. This technique encourages reflection from both opposing positions: the attacker, so they don't trust everything they try to open, and the defender, in implementing effective detection and response against attacks. To implement the honeypot, it was very important to thoroughly understand how to create a persistent service in the operating system and ensure it remained hidden with the appropriate permissions configured. Then, thanks to the good performance of Audit-Tool, the hidden binary with the SUID bit enabled can be located, demonstrating that a simple misconfiguration can compromise the entire system.

The last tool developed was defense_tool, which allows us to detect in real time any attempt to access sensitive files, thus providing great robustness in our detection and response. This tool is important both for protecting the system and for the proper development of hackathons, as it allows organizers to block access to sensitive system files that participants should not access and to raise alerts about it. For its development, we based it on the implementation of monitoring_tool, since the operating logic is very similar. It also acts as a service, so it is present in the operating system once it starts up. All of this was developed in the same environment as the one present on the computers in the laboratories of the School of Computer Science, and therefore, it is fully replicable, forming part of our future work.

All of this reflects the importance of implementing continuous monitoring and improving cybersecurity awareness to adopt better practices for protecting systems. Since even a simple misconfiguration can compromise the entire system, both the solution and the problems and errors lie with the people. That is why we conclude that it is essential that all individuals involved in the development, administration, or use of a computer system are aware that even a small mistake can become a serious error or a critical security breach. Cybersecurity does not depend solely on technological tools but also on the knowledge and good practices of every person who uses a computer system.

Finally, we must highlight the importance of having taken the subjects Networks and Security I and II, Networks, Operating Systems, Advanced Operating Systems and Networks, IT Auditing I and II, and Configuration Assessment, which provided us with a solid foundation to take the first steps of the project and allowed us to delve deeper into different aspects, giving us a broader view of the problems and solutions that exist today in the world of cybersecurity.

Future Work

The creation of this environment is just the beginning of what could become an even larger project. All the work and the environment are focused on its future implementation in the laboratories of the School of Computer Science; therefore, our next steps would head in that direction, transferring all the tools and configurations. This replication of the

work carried out would be very simple due to the fact that the operating system used is the same. After having validated our hackathon proposal, as we saw in the previous chapter, we are motivated to organize an official in-person event, with a more complex design and a longer planning period. However, among the upcoming challenges are obtaining the approval of the School, attracting sponsors, and coordinating resources on a larger scale.

Although we initially conceived this initiative as a small-scale pilot, its potential goes beyond the academic scope. We envision the organization of corporate hackathons gathering hundreds of participants, even turning it into a business opportunity. The three members of the group have taken the course on Business Creation with Professor David Pascual, and since then we have considered turning this experience into an entrepreneurial initiative. Therefore, during the summer holidays we will evaluate whether our enthusiasm translates into a viable business model or if, on the contrary, it has simply been the satisfaction of completing a great challenge.

BIBLIOGRAFÍA

[1] «¿Qué es Scrum?,» [En línea]. Available: https://asana.com/es/resources/what-is-scrum.

[2] «Design Thinking España,» [En línea]. Available: https://xn--designthinkingespaa-d4b.com/%E2%96%B7-que-es-un-hackatón-soluciones-innovadoras#:~:text=Aunque%20el%20origen%20de%20los,diferentes%20tem%C3%A1ticas%2C%20sectores%20y%20p%C3%BAblicos..

[3] «OpenBSD Hackathons: OpenBSD.org,» [En línea]. Available: https://www.openbsd.org/hackathons.html.

[4] «What is a hackathon? A brief history: info.devpost.com,» [En línea]. Available: https://info.devpost.com/blog/what-is-a-hackathon.

[5] «Capture the flag (cybersecurity),» [En línea]. Available: https://en.wikipedia.org/wiki/Capture_the_flag_(cybersecurity) .

[6] «Pwn2Own,» [En línea]. Available: https://en.wikipedia.org/wiki/Pwn2Own .

[7] «DEF CON,» [En línea]. Available: https://en.wikipedia.org/wiki/DEF_CON.

[8] «Capture The Flag with Google,» [En línea]. Available: https://capturetheflag.withgoogle.com/ .

[9] «9 Essential Cyber Security Tools and Techniques,» [En línea]. Available: https://www.devry.edu/blog/cyber-security-tools-and-techniques.html.

[10] «What is an intrusion detection system (IDS)?,» [En línea]. Available: https://www.ibm.com/think/topics/intrusion-detection-system?.

[1] «Top 10 Offensive Security Tools for 2025,» [En línea]. Available: https://scytale.ai/resources/top-offensive-security-tools/.

[2] «Metasploit Documentation: Rapid7 Docs,» [En línea]. Available: https://docs.rapid7.com/metasploit/.

[3] «Suricata User Guide: Open Information Security Foundation,» [En línea]. Available: https://suricata.io/features/.

[4] «Snort 3 User Manual: Snort.org,» [En línea]. Available: https://docs.snort.org.

[5] «Cisco Identity Services Engine (ISE) Data Sheet: Cisco.com,» [En línea]. Available: https://www.cisco.com/c/en/us/products/security/identity-services-engine/index.html.

[6] «PacketFence Documentation: PacketFence.org,» [En línea]. Available: https://packetfence.org/documentation/.

[7] «Zeek Documentation: Zeek.org,» [En línea]. Available: https://docs.zeek.org/.

[8] «Wireshark User's Guide: Wireshark.org,» [En línea]. Available: https://www.wireshark.org/docs/wsug_html/.

[9] «Nagios Core Documentation: Nagios.org,» [En línea]. Available: https://library.nagios.com/products/nagios-core/documentation/.

[10] «Elastic Stack Documentation: Elastic.co,» [En línea]. Available: https://www.elastic.co/guide/index.html.

[11] «Wazuh Documentation: Documentation.Wazuh.com,» [En línea]. Available: https://documentation.wazuh.com/.

[2 «Splunk Documentation: Docs.Splunk.com,» [En línea]. Available:
2] https://docs.splunk.com/Documentation.

[2 «Dionaea Documentation: Read the Docs,» [En línea]. Available:
3] https://dionaea.readthedocs.io/.

[2 «Honeyd Documentation: Honeyd.org,» [En línea]. Available:
4] http://www.honeyd.org/.

[2 «Canarytokens Documentation: Canarytokens.org,» [En línea]. Available:
5] https://docs.canarytokens.org/.

[2 «Bur Suite Documentation: PortSwigger,» [En línea]. Available:
6] https://portswigger.net/burp/documentation.

[2 «Nmap Network Scanning: Nmap.org,» [En línea]. Available:
7] https://nmap.org/docs.html.

[2 «Resources: Cymulate,» [En línea]. Available: https://cymulate.com/resources/.
8]

[2 «Resources: SafeBreach,» [En línea]. Available:
9] https://www.safebreach.com/resources/.

[3 «Autopsy Documentation: Sleuth Kit,» [En línea]. Available:
0] https://www.sleuthkit.org/autopsy/docs.php.

[3 «Volatility Foundation,» [En línea]. Available: https://www.volatilityfoundation.org/.
1]

[3 «TestDisk Documentation: CGSecurity,» [En línea]. Available:
2] https://www.cgsecurity.org/wiki/TestDisk.

[3 3] «Recuva Features: CCleaner,» [En línea]. Available: https://www.ccleaner.com/recuva.

[3 4] «Splunk Documentation: Splunk Docs,» [En línea]. Available: https://docs.splunk.com/Documentation/Splunk.

[3 5] «Graylog Documentation: Graylog Docs,» [En línea]. Available: https://docs.graylog.org/.

[3 6] «GNU/Linux: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/GNU/Linux.

[3 7] «Ubuntu 24.04 LTS (Noble Numbat) Release Notes: Ubuntu Discourse,» [En línea]. Available: https://discourse.ubuntu.com/t/ubuntu-24-04-lts-noble-numbat-release-notes/39890.

[3 8] «Distribución de Linux: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Distribución_Linux.

[3 9] «Swappiness: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Swappiness.

[4 0] «Pluggable Authentication Modules para su ejecución en UNIX oLinux,» [En línea]. Available: https://www.ibm.com/docs/es/informix-servers/12.10.0?topic=security-pluggable-authentication-modules-unix-linux.

[4 1] «AppArmor: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/AppArmor.

[4 2] «Qué son los demonios en el concepto Linux,» [En línea]. Available: https://pq.hosting/es/help/que-son-los-demonios-en-el-concepto-linux.

[43] «Address space layout randomization,» [En línea]. Available: https://www.ibm.com/docs/en/zos/2.4.0?topic=overview-address-space-layout-randomization.

[44] «Proteger el grub con contraseña,» [En línea]. Available: https://geekland.eu/proteger-el-grub-con-contrasena/.

[45] «IPv6 address,» [En línea]. Available: https://www.techtarget.com/iotagenda/definition/IPv6-address.

[46] «¿Qué es un servidor DNS?,» [En línea]. Available: https://www.ibm.com/es-es/topics/dns-server.

[47] «Comprobación de puertos abiertos y en escucha en Linux mediante netstat y ss,» [En línea]. Available: https://alexhost.com/es/faq/comprobacion-de-puertos-abiertos-y-en-escucha-en-linux-mediante-netstat-y-ss/.

[48] «Definición de Modo Promiscuo,» [En línea]. Available: https://www.vpnunlimited.com/es/help/cybersecurity/promiscuous-mode?srsltid=AfmBOope1DW3oGLUwEp6T5ezQnq4AzltqbQjVxcEgOdhlx_oFas06ZcS.

[49] «¿Qué es el firewall UFW y cómo configurarlo en Linux?,» [En línea]. Available: https://www.swhosting.com/es/comunidad/manual/que-es-el-firewall-ufw-y-como-configurarlo-en-linux.

[50] «¿Qué es una conexión VPN, para qué sirve y qué ventajas tiene?,» [En línea]. Available: https://www.xataka.com/basics/que-es-una-conexion-vpn-para-que-sirve-y-que-ventajas-tiene.

[51] «Understanding /etc/passwd File Format,» [En línea]. Available: https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/.

[52] «Algoritmos hash SHA-1 vs SHA-2 vs SHA-256 vs SHA-512,» [En línea]. Available: https://www.ssldragon.com/es/blog/sha1-sha2-sha256-sha-512/#what-is-sha.

[53] «¿Qué es Setgid?,» [En línea]. Available: https://keepcoding.io/blog/que-es-setgid/.

[54] «¿Qué es Setuid?,» [En línea]. Available: https://keepcoding.io/blog/que-es-setuid/.

[55] «What is Apache? In-Depth Overview of Apache Web Server,» [En línea]. Available: https://www.sumologic.com/blog/apache-web-server-introduction.

[56] «What Is NGINX?,» [En línea]. Available: https://www.papertrail.com/solution/guides/nginx/.

[57] «MySQL: Understanding What It Is and How It's Used,» [En línea]. Available: https://www.oracle.com/mysql/what-is-mysql/.

[58] «Bash history y autocompletado: Navegar por el historial de comandos,» [En línea]. Available: https://apuntes.de/linux-certificacion-lpi/el-bash-history-y-el-autocompletado/#gsc.tab=0.

[59] «Capítulo 8. Cifrado de dispositivos de bloque mediante LUKS,» [En línea]. Available: https://docs.redhat.com/es/documentation/red_hat_enterprise_linux/8/html/security_hardening/encrypting-block-devices-using-luks_security-hardening.

[60] «Extensión de archivo .log: todo acerca de los archivos de registro,» [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/extension-de-archivo-log/.

[61] «Infection Monkey: GitHub (Guardicore),» [En línea]. Available: https://github.com/guardicore/monkey.

[62] «Sistema operativo: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_operativo.

[63] «About Ubuntu: Ubuntu.com,» [En línea]. Available: https://ubuntu.com/about.

[64] «Espacio de intercambio: Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Espacio_de_intercambio.

[65] «Package Management: Ubuntu Documentation,» [En línea]. Available: https://documentation.ubuntu.com/server/.

[66] «Python Documentation: Python.org,» [En línea]. Available: https://www.python.org/doc/.

[67] «Scapy Documentation: Scapy Project,» [En línea]. Available: https://scapy.net.

[68] «Scrapy Documentation: Scrapy Team,» [En línea]. Available: https://docs.scrapy.org/.

[69] «Pyperclip Documentation: PyPI,» [En línea]. Available: https://pypi.org/project/pyperclip/.

[70] «python-nmap Documentation: PyPI,» [En línea]. Available: https://pypi.org/project/python-nmap/.

[71] «Beautiful Soup Documentation: crummy.com,» [En línea]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[72] «Requests Documentation: Requests Library,» [En línea]. Available: https://docs.python-requests.org/en/latest/.

[7 «Impacket: Network Protocol Library: SecureAuthCorp GitHub,» [En línea]. Available:
3] https://github.com/SecureAuthCorp/impacket.

[7 «PyCryptodome Documentation: Read the Docs,» [En línea]. Available:
4] https://pycryptodome.readthedocs.io/.

[7 «Scikit-learn Documentation: Scikit-learn.org,» [En línea]. Available: https://scikit-
5] learn.org/stable/.

APÉNDICES

Apéndice A - Panfleto del Hackatón de Ciberseguridad

Objetivo del Hackatón

Tu misión es infiltrarte en una máquina virtual, encontrar una vulnerabilidad del sistema y usarla para acceder a información confidencial. Para ello, deberás utilizar tus conocimientos sobre sistemas Linux, permisos especiales (como el bit SUID) y habilidades básicas de análisis y explotación.

Requisitos Previos

Antes de comenzar, asegúrate de estudiar diferentes ámbitos de ciberseguridad, centrándote especialmente en los relacionados con la escalada de privilegios. Con los conocimientos adquiridos en las asignaturas Redes y Seguridad 1 y Redes y Seguridad 2 deberías estar casi preparado para afrontar el reto.

¿Qué Aprenderás?

- Localización y análisis de archivos con permisos especiales (SUID).
- Escalada de privilegios a través de vulnerabilidades mal configuradas.
- Detección de honeypots y análisis de archivos sospechosos.
- Exploración de sistemas de archivos en Linux.

Resolución del Hackatón

Paso 1: Busca Pistas

Explora el sistema en busca de archivos interesantes. El archivo que debe ser encontrado actúa como un honeypot: al leerlo, se activará una alerta para los organizadores. Así sabremos que has caído en la trampa… o que vas por buen camino.

Paso 2: Encuentra el ejecutable vulnerable

Investiga y encuentra un comando que sirva para localizar archivos con el bit SUID activado. Busca entre los resultados alguno especialmente sospechoso.

Paso 3: Explota la vulnerabilidad

Ejecuta el binario oculto. Introduce la contraseña descubierta en el honeypot. Si todo va bien, la caja fuerte digital se abrirá y podrás ver el contenido de secretos_del_banco.txt.

¿Cómo funciona internamente la caja fuerte?

El binario oculto tiene permisos especiales que permiten ejecutar su código con permisos de root. Este binario te pide una contraseña, y si aciertas, te muestra el contenido de un archivo protegido.

Notas Finales

- No es necesario tener acceso root para superar el reto.
- El objetivo es demostrar tus habilidades de exploración, análisis y lógica.
- Las alertas generadas al interactuar con archivos clave serán usadas para evaluar tu progreso

¡IMPORTANTE!

No modifiques configuraciones del sistema fuera del entorno proporcionado. Este reto es auto-contenido y supervisado. Disfruta la experiencia y aprende al máximo.

¿Listo para el desafío?

Ponte tu máscara de hacker ético y... ¡que comience el juego!

Apéndice B - Bases del Hackatón

Objetivo general

El presente hackatón tiene como finalidad poner a prueba los conocimientos prácticos de los participantes en ciberseguridad ofensiva y defensiva, dentro de un entorno controlado. Los asistentes deberán identificar y explotar vulnerabilidades en una máquina virtual configurada intencionadamente con debilidades de seguridad, aplicando técnicas de exploración, escalada de privilegios y análisis forense.

Participantes

Podrán participar estudiantes de grado en Ingeniería Informática u otras titulaciones afines, con conocimientos básicos en sistemas Linux, redes y seguridad informática y matriculados en la Universidad Complutense de Madrid. Se recomienda haber superado previamente asignaturas como Redes y Seguridad 1 y Redes y Seguridad 2.

- Modalidad: Individual o por equipos de 2 personas
- Plazas disponibles: 5 equipos o participantes.

Desarrollo del Reto

Los participantes tendrán acceso a una máquina virtual vulnerable en un entorno cerrado. El reto se compone de varias fases:

- **Exploración inicial:** localización de archivos y servicios sospechosos.
- **Detección de honeypots** y activadores de alertas.
- **Identificación de archivos con bit SUID** u otras configuraciones de riesgo.
- **Escalada de privilegios** mediante explotación controlada de binarios vulnerables.
- **Acceso a contenido confidencial** como prueba de éxito.

Todas las acciones realizadas son monitorizadas en tiempo real mediante herramientas desarrolladas por el equipo organizador para evaluar la evolución del reto.

Evaluación

El progreso de los participantes será evaluado en función de:

- Completitud del reto técnico (acceso a secretos_del_banco.txt).
- Activación de eventos monitorizados (lectura de honeypots, ejecución de binarios clave).
- Tiempo de resolución.
- Análisis post-mortem (opcional): explicación del proceso seguido y lecciones aprendidas.

El participante o equipo que logre superar el reto en el menor tiempo posible y demuestre mayor conocimiento sobre los contenidos que se tratan, obtendrán un mes de subscripción en la plataforma TryHackMe.

Normas del Evento

- No se permite modificar configuraciones del sistema fuera del entorno provisto.
- No se autoriza ningún tipo de ataque entre participantes ni hacia la infraestructura.
- Las herramientas externas están permitidas, siempre que no vulneren el entorno.
- El reto es auto-contenido y será supervisado continuamente por el equipo docente.

Cualquier infracción grave de estas normas implicará la descalificación inmediata.

Herramientas y Recursos

Durante el evento, se utilizarán tecnologías como:

- Sistema operativo: Ubuntu 24.04 LTS
- Auditoría en tiempo real: auditd, audit_tool.py, monitoring_tool.py
- Análisis de privilegios: comandos nativos (find, ls, getcap, etc.)
- Visualización de eventos en tiempo real para el comité organizador

El entorno de pruebas ha sido preparado con vulnerabilidades específicas, documentadas y controladas por el equipo organizador para garantizar una experiencia segura y educativa.

Fecha, Duración y Lugar

- Fecha del evento: dd/mm/aaaa
- Duración aproximada: 3 horas
- Lugar: Laboratorio XXX, Facultad de Informática, UCM

Inscripción

La inscripción deberá realizarse antes del [fecha límite] a través del siguiente formulario: [enlace a formulario]

Protección de datos personales y consentimiento informado

Este hackatón está organizado por la Facultad de Informática de la Universidad Complutense de Madrid, con fines exclusivamente formativos y académicos.

Durante el desarrollo del evento:

- Se podrán recoger datos técnicos relacionados con la participación (como eventos generados al interactuar con archivos específicos o ejecutar acciones supervisadas), con el único propósito de evaluar el progreso, generar informes estadísticos internos y contribuir a la mejora del entorno académico.
- Los datos personales facilitados en la inscripción (nombre, correo institucional, grupo o titulación) serán tratados de acuerdo con la normativa vigente, exclusivamente para gestionar la participación en el evento.
- No se recogen ni almacenan datos personales fuera del ámbito técnico y académico descrito. En ningún caso se cederán a terceros.

La participación en el hackatón implica la aceptación tácita de estas condiciones.

Para más información o para ejercer los derechos recogidos en el Reglamento General de Protección de Datos (UE 2016/679), puedes contactar con el responsable del tratamiento de datos en la Facultad de Informática a través del correo: [email institucional del profesor o responsable del evento].

Apéndice C - Presentación Hackatón

Objetivo del Hackatón de Ciberseguridad

El reto consiste en la exploración de una máquina virtual con el fin de identificar y explotar vulnerabilidades del sistema que permitan obtener acceso a información confidencial. Para ello, se emplearán conocimientos avanzados en entornos Linux, con especial atención al manejo de permisos especiales (bit SUID) y al uso de técnicas de análisis y explotación propias de pruebas de penetración.

Requisitos Previos para el Hackatón

Antes de iniciar la competición, se recomienda al participante:

- Revisar los fundamentos de ciberseguridad, haciendo hincapié en la escalada de privilegios.
- Repasar los contenidos de las asignaturas de Redes y Seguridad I y II.
- Familiarizarse con las herramientas básicas de pentesting para la exploración y análisis de sistemas.

Explorando la Ciberseguridad (Qué aprenderás)

Durante esta fase, el participante desarrollará competencias en:

- **Localización y análisis de archivos con bit SUID:** identificación de ejecutables con permisos especiales y valoración de su riesgo.
- **Escalada de privilegios:** aprovechamiento de configuraciones erróneas para obtener permisos elevados.
- **Detección de honeypots:** reconocimiento de archivos y sistemas trampa diseñados para alertar a los organizadores.
- **Exploración de sistemas de archivos en Linux**: uso de técnicas de navegación y mapeo para descubrir rutas y directorios relevantes

Resolviendo el Hackatón

- *Paso 1: Búsqueda de pistas*

Se debe rastrear el sistema hasta encontrar el fichero denominado “honeypot”. La lectura de este archivo disparará una alerta para los organizadores y validará el hallazgo.

- *Paso 2: Identificación del binario vulnerable*
  Ejecución del comando: find / -perm -4000 2>/dev/null
  con el fin de listar los ejecutables que posean el bit SUID activado. De entre ellos, se seleccionará el más relevante según el contexto.
- *Paso 3: Explotación y acceso*
  Ejecución del binario identificado e introducción de la contraseña descubierta en el honeypot para acceder al archivo protegido, concluyendo así el reto.

Detalles Técnicos del Binario Oculto

El binario seleccionado posee permisos especiales que le permiten ejecutarse con privilegios de root sin necesidad de él; solicita una contraseña al usuario y, en caso de ser correcta, muestra el contenido de un archivo resguardado. Este mecanismo facilita la demostración de la escalada de privilegios y la correcta explotación de la vulnerabilidad.

Notas Finales del Reto

- No se requiere acceso root para superar el desafío.
- Mantener la integridad del entorno original, evitando modificaciones en configuraciones globales.
- Las alertas generadas a través del honeypot se emplearán como métricas de progreso y evaluación en tiempo real.
- El hackatón está diseñado para ser auto-contenido y supervisado, garantizando la seguridad y la reproducibilidad de las pruebas.

Prepárate para el Desafío de Ciberseguridad

Este hackatón ofrece la oportunidad de demostrar habilidades de hacker ético mediante la resolución de un reto que combina exploración, análisis y explotación de sistemas Linux. Se anima al participante a aprovechar esta experiencia formativa para consolidar conocimientos y técnicas avanzadas.

Apéndice D - Código de Audit-Tool

Se adjunta el enlace al GitHub en el cual se encuentra el código de la herramienta desarrollada:

https://github.com/fedelm8/TFG

Se adjunta la explicación completa del código:

Documentación Audit_Tool

Apéndice E - Código de Monitoring_Tool

Se adjunta el enlace al GitHub en el cual se encuentra el código de la herramienta desarrollada:

https://github.com/fedelm8/TFG2

Se adjunta la explicación completa del código:

Documentación Monitoring_Tool

Apéndice F - Código de Defense-Tool

Se adjunta el enlace al GitHub en el cual se encuentra el código de la herramienta desarrollada:

https://github.com/fedelm8/TFG3

Se adjunta la explicación completa del código:

Documentación Defense_Tool

Apéndice G - Máquinas Virtuales

Se adjuntan los enlaces a las máquinas virtuales en caso de querer probar las herramientas y la resolución del hackatón.

Máquina Virtual Hackatón

Máquina Virtual Herramientas