



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO**

TÍTULO

Análisis de Vulnerabilidades en la Programación de Sockets: Técnicas de
Identificación y Mitigación

AUTOR

CAMATÓN RENDÓN, JOHNNY ALBERTO

TRABAJO DE TITULACIÓN

Previo a la obtención del grado académico en
MAGÍSTER EN CIBERSEGURIDAD

TUTOR

OROZCO IGUASNIA, JAIME BENJAMÍN

Santa Elena, Ecuador

Año 2024



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO
TRIBUNAL DE SUSTENTACIÓN**



Firmado electrónicamente por:
JAIME BENJAMIN
OROZCO IGUASNIA

Ing. Alicia Andrade Vera, Mgtr.
COORDINADORA DEL PROGRAMA

Ing. Jaime Orozco Iguasnia, Mgtr.
TUTOR

Ing. Edison Quintuña Padilla, Mgtr.
DOCENTE ESPECIALISTA

Ing. María Alvarez Galarza, Mgtr.
DOCENTE ESPECIALISTA

Abg. María Rivera González, MSc.
SECRETARIO GENERAL UPSE



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO**

CERTIFICACIÓN

Certifico que luego de haber dirigido científica y técnicamente el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos, razón por el cual apruebo en todas sus partes el presente trabajo de titulación que fue realizado en su totalidad por CAMATÓN RENDÓN JOHNNY ALBERTO, como requerimiento para la obtención del título de Magíster en Ciberseguridad.

TUTOR



firmado electrónicamente por:
**JAIME BENJAMÍN
OROZCO IGUASNIA**

ING. JAIME BENJAMÍN OROZCO IGUASNIA, MGTI.

Santa Elena, 14 de octubre de 2024



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO**

DECLARACIÓN DE RESPONSABILIDAD

Yo, **JOHNNY ALBERTO CAMATÓN RENDÓN**

DECLARO QUE:

El trabajo de Titulación, Análisis de Vulnerabilidades en la Programación de Sockets: Técnicas de Identificación y Mitigación previo a la obtención del título en Magíster en Ciberseguridad, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

Santa Elena, 14 de octubre de 2024

EL AUTOR

JOHNNY ALBERTO CAMATÓN RENDÓN Firmado digitalmente
por JOHNNY ALBERTO
CAMATÓN RENDÓN


JOHNNY ALBERTO CAMATÓN RENDÓN



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO**

CERTIFICACIÓN DE ANTIPLAGIO

Certifico que después de revisar el documento final del trabajo de titulación denominado Análisis de Vulnerabilidades en la Programación de Sockets: Técnicas de Identificación y Mitigación, presentado por el estudiante, CAMATÓN RENDÓN JOHNNY ALBERTO fue enviado al Sistema Antiplagio COMPILATIO, presentando un porcentaje de similitud correspondiente al 1%, por lo que se aprueba el trabajo para que continúe con el proceso de titulación.

 CERTIFICADO DE ANÁLISIS
magister

**JOHNNY ALBERTO CAMATÓN
RENDÓN**

1%
Textos
sospechosos

0% Similitudes
0% similitudes entre comillas
0% entre las fuentes
mencionadas
1% Idiomas no reconocidos

Nombre del documento: JOHNNY ALBERTO CAMATÓN RENDÓN.pdf ID del documento: 2199025a77a03146e9e5a7c7d3c3857ddd8510b Tamaño del documento original: 1,19 MB Autores: []	Depositante: JAIME BENJAMÍN OROZCO IGUASNIA Fecha de depósito: 13/10/2024 Tipo de carga: interface fecha de fin de análisis: 13/10/2024	Número de palabras: 8469 Número de caracteres: 59.385
---	--	--

Ubicación de las similitudes en el documento:

TUTOR



Firmado electrónicamente por:
**JAIME BENJAMÍN
OROZCO IGUASNIA**

ING. JAIME BENJAMÍN OROZCO IGUASNIA, MGTI.



**UNIVERSIDAD ESTATAL PENÍNSULA
DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
INSTITUTO DE POSTGRADO**

AUTORIZACIÓN

Yo, JOHNNYALBERTO CAMATÓN RENDÓN

Autorizo a la Universidad Estatal Península de Santa Elena, para que haga de este trabajo de titulación o parte de él, un documento disponible para su lectura consulta y procesos de investigación, según las normas de la Institución.

Cedo los derechos en línea patrimoniales de mi propuesta metodológica y tecnológica avanzada con fines de difusión pública, además apruebo la reproducción de esta propuesta metodológica y tecnológica avanzada dentro de las regulaciones de la Universidad, siempre y cuando esta reproducción no suponga una ganancia económica y se realice respetando mis derechos de autor

Santa Elena, 14 de octubre de 2024

EL AUTOR

JOHNNY ALBERTO Firmado digitalmente
CAMATON RENDON por JOHNNY ALBERTO
CAMATON RENDON

JOHNNY ALBERTO CAMATÓN RENDÓN

AGRADECIMIENTO

Mi más profundo agradecimiento a Dios, cuya guía y fortaleza han sido constantes en cada etapa de este camino académico.

A mis queridos padres, expreso mi eterna gratitud por su amor incondicional, sacrificios y apoyo inquebrantable. Sus enseñanzas y aliento han sido fundamentales en cada paso de este logro.

Con cariño, agradezco a mi novia por ser un pilar emocional invaluable, brindándome comprensión y motivación durante todo este proceso.

Un reconocimiento especial al Ing. Orozco Jaime, Mgti., por su dedicada tutoría. Su experiencia y orientación han sido cruciales para el éxito de este proyecto.

Extiendo también mi sincero agradecimiento al Ing. Andrés Tomalá Mgtr., cuya valiosa contribución y conocimientos han enriquecido significativamente este trabajo.

A todos ustedes, mi más sincero agradecimiento. Sus diversas contribuciones han sido esenciales en mi desarrollo académico y personal.

Gracias a todos.

Johnny Alberto Camatón Rendón

DEDICATORIA

En humilde reconocimiento, ofrezco este trabajo a DIOS todopoderoso, cuya luz ha iluminado mi sendero académico, otorgándome la determinación para alcanzar esta meta.

Con amor, dedico este logro a mis padres, Johnny Camatón Arizábal y Lourdes Rendón Buste, y a mis hermanas, Genesis y Danna. Su apoyo incondicional y cariño han sido fundamentales en este viaje. Este trabajo es testimonio de mi gratitud hacia ustedes, mi preciada familia.

Johnny Alberto Camatón Rendón

ÍNDICE GENERAL

TÍTULO.....	I
TRIBUNAL DE SUSTENTACIÓN	II
CERTIFICACIÓN.....	III
DECLARACIÓN DE RESPONSABILIDAD	IV
DECLARO QUE:	IV
CERTIFICACIÓN DE ANTIPLAGIO	V
AUTORIZACIÓN	VI
AGRADECIMIENTO	VII
DEDICATORIA.....	VIII
ÍNDICE GENERAL	IX
ÍNDICE DE TABLAS	XII
ÍNDICE DE FIGURAS	XIII
RESUMEN	XIV
ABSTRACT.....	XV
INTRODUCCIÓN	1
CAPÍTULO 1.	5
1. MARCO TEÓRICO REFERENCIAL.....	5
1.1. Revisión de literatura	5
1.2. Desarrollo teórico y conceptual	6
1.2.1 Fundamentos de la programación de sockets	6
1.2.2 Tipos básicos de sockets	7
1.2.3 Modelos de programación de sockets	7
1.2.4 Conceptos básicos de seguridad en sockets.....	8

1.2.5 Vulnerabilidades comunes en la programación de sockets	8
1.2.6 Prácticas básicas de seguridad en sockets.....	9
1.2.7 Protocolos de seguridad comunes.....	10
1.2.8 Técnicas básicas de detección de intrusiones	10
1.2.9 Pruebas de seguridad para aplicaciones basadas en sockets.....	11
CAPÍTULO 2. METODOLOGÍA.....	12
2.1. Contexto de la investigación	12
2.2. Diseño y alcance de la investigación	13
2.3. Tipo y métodos de investigación.....	14
2.4. Población y muestra	14
2.5. Metodología	15
2.5.1. Configuración de entorno en Proxmox.....	15
2.5.2. Selección y despliegue de aplicaciones	21
2.5.3. Despliegue de aplicaciones.....	23
2.5.4. Justificación de la selección.....	24
2.5.5. Evaluación inicial de vulnerabilidades	25
2.5.6. Aplicaciones de técnicas de seguridad.....	28
2.5.7. Análisis comparativo	29
CAPÍTULO 3. RESULTADOS Y DISCUSIÓN	32
3.1. Categorización y evolución de vulnerabilidades.....	32
3.2. Efectividad de las técnicas de análisis	32
3.3. Implementación de mejoras de seguridad	34
3.3.1. Aplicación de Chat Simple	34
3.3.2. Aplicación de Dibujo Colaborativo	35
3.4. Análisis comparativo pre y post implementación	35
3.5. Marco de referencia propuesto.....	36

CONCLUSIONES	39
RECOMENDACIONES	41
REFERENCIAS.....	42
ANEXOS.....	46

ÍNDICE DE TABLAS

Tabla 1. Detalles de la aplicación de chat.....	22
Tabla 2. Detalles de la aplicación de dibujo colaborativo	23
Tabla 3. Resultados del escaneo automatizado de las aplicaciones.....	26
Tabla 4. Resultados del escaneo de las aplicaciones	29
Tabla 5. Tabla Comparativa de Técnicas de Análisis.....	33

ÍNDICE DE FIGURAS

Figura 1. Mapa Espacial de Santa Elena.....	12
Figura 2. Página de oficial de Proxmox.....	16
Figura 3. Pantalla de instalación de PVE.....	17
Figura 4. Página de descarga de Ubuntu 22.04.....	18
Figura 5. Interfaz de gestión de PVE.....	19
Figura 6. Creación de máquinas virtuales.....	19
Figura 7. Pagina inicial de Proxmox.....	20
Figura 8. Configuración de SSH para los servidores.....	21
Figura 9. Comandos para el despliegue de la aplicación de chat	23
Figura 10. Comandos para el despliegue de aplicación de pizarra.....	24
Figura 11. Resultados del escaneo de vulnerabilidades con OWASP ZAP	26
Figura 12. Resultados del escaneo de red en Wireshark.....	27
Figura 13. Gráfico de Radar de Técnicas de Análisis.....	34
Figura 14. Gráfico comparativo pre y post implementación	35
Figura 15. Marco de referencia propuesta	36

RESUMEN

Esta investigación aborda el crítico problema de las vulnerabilidades en la programación de sockets, centrándose en su identificación y mitigación efectiva. El objetivo principal es desarrollar un marco integral que combine técnicas avanzadas de análisis de código y mejores prácticas de desarrollo para fortalecer la seguridad de las aplicaciones de red. Utilizando una metodología mixta, se analizan aplicaciones de código abierto para categorizar vulnerabilidades comunes, evaluar técnicas de detección y proponer estrategias de mitigación. Los resultados revelan patrones recurrentes de vulnerabilidades y la eficacia de enfoques combinados para su detección. Se concluye que la integración de análisis estático, dinámico y prácticas de codificación segura mejora significativamente la robustez de las aplicaciones basadas en sockets frente a amenazas cibernéticas. Los resultados ofrecen una guía práctica para que desarrolladores y empresas implementen medidas de seguridad en el ciclo de desarrollo de software, permitiendo identificar y mitigar vulnerabilidades de manera proactiva.

Palabras claves: Seguridad de sockets, Análisis de vulnerabilidades, Programación segura.

ABSTRACT

This research addresses the critical issue of vulnerabilities in socket programming, focusing on their effective identification and mitigation. The main objective is to develop a comprehensive framework that combines advanced code analysis techniques and best development practices to strengthen the security of network applications. Using a mixed methodology, open-source applications are analyzed to categorize common vulnerabilities, evaluate detection techniques, and propose mitigation strategies. The results reveal recurring vulnerability patterns and the effectiveness of combined approaches for their detection. It is concluded that the integration of static and dynamic analysis, along with secure coding practices, significantly improves the robustness of socket-based applications against cyber threats. The results provide a practical guide for developers and companies to implement security measures in the software development lifecycle, allowing proactive identification and mitigation of vulnerabilities.

Keywords: Socket security, Vulnerability analysis, Secure programming

INTRODUCCIÓN

En la era actual de internet, las aplicaciones web y servicios en línea se han convertido en componentes fundamentales de nuestra infraestructura digital. En este contexto, la seguridad de la información ha emergido como una preocupación crítica, especialmente en lo que respecta a la comunicación entre clientes y servidores web. Los sockets TCP, como elementos esenciales de esta comunicación, juegan un papel crucial en la arquitectura de las aplicaciones web modernas. Sin embargo, si no se implementan o utilizan correctamente, estos sockets pueden albergar vulnerabilidades significativas que exponen las aplicaciones web a una variedad de ataques cibernéticos.

La presencia de vulnerabilidades relacionadas con los sockets TCP en las aplicaciones web ha sido extensamente documentada en el ámbito de la seguridad informática. Casos destacados, tales como los ataques de inyección SQL mediante entradas de formularios web mal sanitizadas (Clarke, 2009) y los ataques de denegación de servicio (DoS) que aprovechan la gestión incorrecta de conexiones TCP (Zargar, 2013), han evidenciado las graves repercusiones que pueden surgir de las debilidades en la implementación de sockets TCP en entornos web. Estos incidentes no sólo han provocado pérdidas económicas significativas, sino que también han debilitado la confianza de los usuarios en la seguridad de las aplicaciones web.

Pese al aumento de la conciencia acerca de la relevancia de la seguridad en la programación de sockets TCP para aplicaciones web, las vulnerabilidades continúan siendo sorprendentemente habituales. Una reciente investigación de (OWASP, 2021) reveló que las fallas de seguridad asociadas con la administración de sesiones y la validación de entrada, ambas estrechamente vinculadas a la implementación de sockets TCP, continúan siendo las diez vulnerabilidades más destacadas de las aplicaciones web.

La detección efectiva y la mitigación de las vulnerabilidades en la programación de sockets TCP para aplicaciones web representan desafíos considerables para la comunidad de desarrollo de software. Aunque hay instrumentos automatizados, tales como los escáneres de vulnerabilidades en la web, estos frecuentemente no poseen la sofisticación requerida para identificar problemas complejos o sutiles en la implementación de sockets

TCP (Doupé, 2010). Adicionalmente, el ritmo vertiginoso del desarrollo de aplicaciones web contemporáneas, propulsado por metodologías ágiles y DevOps, frecuentemente limita el tiempo para un análisis de seguridad exhaustivo. En consecuencia, numerosos desarrolladores web no disponen del tiempo, conocimiento o los recursos necesarios para implementar de manera consistente prácticas de codificación segura en sus proyectos de aplicaciones web (Abela, 2021).

En respuesta a estos desafíos, surge la necesidad urgente de desarrollar métodos más sofisticados y holísticos para el análisis, detección y prevención de vulnerabilidades en la programación de sockets TCP para aplicaciones web. Esta investigación tiene como objetivo atender esta necesidad a través de la exploración y evaluación de diversas técnicas de identificación de vulnerabilidades, que incluyen el análisis estático de código, el fuzzing de protocolos web, y la revisión manual de código de aplicaciones en línea. Además, se busca desarrollar un marco integral que combine estas técnicas con las mejores prácticas de codificación segura y los principios de desarrollo de software seguro específicos para entornos web.

La hipótesis central de este trabajo propone que la combinación de diversas metodologías de análisis, junto con la adopción de prácticas de codificación robustas orientadas a la web, puede mejorar significativamente la capacidad de los desarrolladores para identificar y mitigar vulnerabilidades en la programación de sockets TCP en aplicaciones web.

La relevancia de esta investigación se basa en la creciente importancia de la seguridad web en un mundo que se encuentra cada vez más dependiente de las aplicaciones en línea. Con la expansión de los servicios en línea y la ampliación de las arquitecturas de microservicios, la superficie de ataque potencial para explotaciones basadas en sockets TCP en entornos web se encuentra en constante crecimiento (Newman, 2015). Este estudio, al ofrecer instrumentos y metodologías más eficaces para tratar las vulnerabilidades de los sockets TCP en aplicaciones web, tiene el potencial de contribuir de manera significativa a la mejora de la seguridad integral de las aplicaciones en línea.

Además, al centrar la atención en las prácticas de desarrollo web y las metodologías de análisis específicas para entornos web, este estudio tiene como objetivo no solo tratar las

vulnerabilidades presentes, sino también promover un enfoque proactivo hacia la seguridad en el ciclo de vida del desarrollo de aplicaciones web.

Planteamiento de la investigación

La presente investigación se basa en la creciente necesidad de tratar las vulnerabilidades en la programación de sockets, un componente crítico en las aplicaciones de red modernas. La complejidad de los sistemas distribuidos y la continua evolución constante de las amenazas cibernéticas hacen esencial el desarrollo de estrategias más eficaces para la identificación y mitigación de dichas vulnerabilidades.

Esta investigación se justifica por su potencial para mejorar significativamente la seguridad de las aplicaciones de red, protegiendo así la privacidad de los usuarios, reduciendo los riesgos para las organizaciones, y contribuyendo al avance del conocimiento en el campo de la seguridad informática. Además, la investigación busca cubrir una deficiencia en el conocimiento actual, proporcionando un marco integral que combine análisis avanzado de código, mejores prácticas de desarrollo y metodologías de seguridad.

Formulación del problema de investigación

¿Cómo se pueden mejorar las técnicas de identificación y mitigación de vulnerabilidades en la implementación de sockets TCP en aplicaciones web para incrementar la seguridad general de estos sistemas en el contexto del desarrollo de software moderno?

Objetivo General:

Analizar las vulnerabilidades más comunes en la programación de sockets en aplicaciones web y de servidor, evaluando su impacto y riesgo, y desarrollar un marco integral para su identificación y mitigación efectiva, considerando las últimas tendencias en desarrollo de software.

Objetivos Específicos:

1. Categorizar y evaluar la evolución de las vulnerabilidades en la programación de sockets en los últimos 5 años, identificando las tendencias y los riesgos asociados a cada tipo de vulnerabilidad.

2. Comparar la efectividad de técnicas de análisis estático y dinámico para identificar vulnerabilidades en código fuente y binario de aplicaciones que utilizan sockets, considerando factores como la precisión, el rendimiento y la facilidad de uso.
3. Desarrollar un conjunto de mejores prácticas, herramientas y un marco de referencia para la mitigación de vulnerabilidades en la programación de sockets, evaluando su eficacia a través de pruebas de concepto y estudios de caso.

Planteamiento hipotético

La adopción de un enfoque integrado que integre el análisis estático de código, el fuzzing de protocolos web y la revisión manual de código, junto con la adopción de prácticas de codificación segura específicas para entornos web, optimizará notablemente la detección y mitigación de vulnerabilidades en la implementación de sockets TCP en aplicaciones web. Esto contribuirá a una reducción mínima del 50% en la incidencia de vulnerabilidades asociadas con sockets en comparación con las técnicas tradicionales de análisis.

CAPÍTULO 1.

1. MARCO TEÓRICO REFERENCIAL

1.1. Revisión de literatura

En su estudio "A Comprehensive Study on Web Application Vulnerabilities", (Lyu, 2019) destacaron la prevalencia de vulnerabilidades relacionadas con la administración de sesiones y la validación de entrada en aplicaciones web. Estas áreas, estrechamente vinculadas a la implementación de sockets TCP, representan un riesgo constante debido a su complejidad técnica. Pese a los intentos de mejorar la seguridad en estas áreas, los atacantes continúan explotándolas, lo que subraya la necesidad de desarrollar soluciones más avanzadas y eficaces para mitigar estos riesgos en entornos web.

Sharma, en su revisión "A Systematic Review of Web Socket Security Vulnerabilities and Mitigation Techniques" (Sharma, 2020), identificaron vulnerabilidades clave en la tecnología WebSockets, como ataques de intermediario (MITM), inyección de código y ataques de denegación de servicio (DoS). Al aprovechar las debilidades inherentes a los sockets TCP, estas amenazas pueden comprometer la seguridad y estabilidad de las aplicaciones web. Sharma et al. también propusieron un conjunto de prácticas recomendadas para mitigar estos riesgos, que incluyen el uso de cifrado y autenticación robusta, así como controles estrictos de los datos transmitidos.

Chen y colaboradores en "Dynamic Analysis of TCP Socket Vulnerabilities in Modern Web Applications" (Chen, 2022), presentaron un enfoque dinámico para detectar vulnerabilidades en aplicaciones web mediante el monitoreo del tráfico de red en tiempo real. A través de esta metodología, descubrieron vulnerabilidades que los análisis estáticos convencionales no lograban identificar, especialmente en aplicaciones con lógica de negocio compleja. El análisis dinámico de Chen et al. destacó la importancia de implementar métodos de seguridad más avanzados en la etapa de desarrollo para proteger mejor las aplicaciones web frente a amenazas emergentes.

Gupta y Singh, en "Comparative Analysis of Socket Security in RESTful and GraphQL APIs" (Gupta, 2023), compararon las vulnerabilidades de seguridad entre las APIs

RESTful y GraphQL. Descubrieron que, aunque las APIs GraphQL ofrecen mayor flexibilidad, también presentan nuevos desafíos de seguridad en la implementación de sockets TCP. Estos desafíos incluyen la validación adecuada de consultas y la mitigación de ataques de denegación de servicio (DoS), que requieren una atención especial para garantizar la seguridad en entornos GraphQL, frente a las APIs RESTful, que ya cuentan con medidas de seguridad más establecidas.

(Nakamura, 2021), en su trabajo "Fuzzing Socket APIs for Uncovering Vulnerabilities in Web Servers", utilizaron una técnica de fuzzing específica para descubrir vulnerabilidades en las implementaciones de sockets TCP en servidores web. Su investigación reveló vulnerabilidades críticas en servidores populares, que no habían sido identificadas previamente. Este trabajo destacó la importancia de realizar pruebas exhaustivas y constantes en los servidores, utilizando técnicas como el fuzzing, para detectar y mitigar posibles fallos de seguridad antes de que puedan ser explotados por atacantes.

Estas investigaciones subrayan la constante evolución de las amenazas de seguridad en las aplicaciones web que emplean sockets TCP. Desde vulnerabilidades fundamentales como las inyecciones SQL hasta desafíos más sofisticados en las implementaciones de WebSocket y GraphQL, la literatura destaca la necesidad de un enfoque multifacético para abordar la seguridad. La inclinación hacia el análisis dinámico y las técnicas de fuzzing evidencia la sofisticación en aumento tanto de los ataques como de las estrategias de defensa.

1.2. Desarrollo teórico y conceptual

1.2.1 Fundamentos de la programación de sockets

Los sockets son una tecnología fundamental en las comunicaciones de red, permitiendo que diferentes programas intercambien datos a través de una red. Según (Tanenbaum & Wetherall, 2011) los sockets actúan como puntos finales de comunicación bidireccional entre dos programas que se ejecutan en una red.

En el contexto de las aplicaciones web modernas, los sockets TCP son fundamentales para establecer conexiones persistentes, crucial para funcionalidades como chats en

tiempo real o actualizaciones en vivo. Según (Fielding, 2014), los sockets TCP subyacen en protocolos como HTTP/1.1, formando la base de la comunicación web.

1.2.2 Tipos básicos de sockets

1. Sockets TCP: Proporcionan una comunicación orientada a la conexión y confiable. (Stevens, Fenner, & Rudoff, 2004) explican que TCP garantiza la entrega ordenada y sin errores de los datos.
2. Sockets UDP: Ofrecen comunicación sin conexión, más rápida pero menos confiable. Según (Comer, 2000), UDP es útil para aplicaciones que priorizan la velocidad sobre la fiabilidad.
3. Sockets Raw: Permiten acceso directo a protocolos de red de bajo nivel. (Parziale, y otros, 2006) indican que estos son útiles para desarrollar nuevos protocolos o acceder a características específicas de protocolos existentes.
4. WebSockets: Introducidos con HTML5, los WebSockets permiten comunicación bidireccional full-dúplex sobre una única conexión TCP. (Fette, 2011) explican que los WebSockets son especialmente útiles para aplicaciones web que requieren actualizaciones en tiempo real.

1.2.3 Modelos de programación de sockets

1. Bloqueante: El programa se detiene hasta que se complete la operación de E/S.
2. No bloqueante: El programa continúa ejecutándose mientras se realiza la operación de E/S.
3. Multiplexado: Permite manejar múltiples conexiones simultáneamente.
4. Asíncrono: Utiliza callbacks o eventos para manejar operaciones de E/S.
5. Reactivo: Un modelo emergente que utiliza programación asíncrona y flujos de datos para manejar grandes volúmenes de conexiones concurrentes. (Boner, 2014) argumentan que este modelo es particularmente adecuado para aplicaciones web escalables.

(Schmidt, Stal, Rohnert, & Buschmann, 2000) discuten estos modelos en detalle, destacando sus ventajas y desventajas en diferentes escenarios de aplicación.

1.2.4 Conceptos básicos de seguridad en sockets

La seguridad en la programación de sockets es crucial para proteger la integridad, confidencialidad y disponibilidad de los datos transmitidos. (Stallings, 2017) destaca varios conceptos clave:

1. Confidencialidad: Asegurar que solo los destinatarios autorizados puedan leer los datos transmitidos.
2. Integridad: Garantizar que los datos no sean alterados durante la transmisión.
3. Autenticación: Verificar la identidad de las partes involucradas en la comunicación.
4. Disponibilidad: Asegurar que los servicios de red estén accesibles cuando se necesiten.
5. No repudio: Garantizar que una parte no pueda negar haber enviado o recibido un mensaje.
6. Principio de mínimo privilegio: (Saltzer & Schroeder, 1975) introdujeron este principio, que en el contexto de sockets implica limitar los permisos y el acceso de las conexiones a solo lo estrictamente necesario.

(Anderson, 2008) añade que estos conceptos deben considerarse en conjunto para lograr una seguridad efectiva en las aplicaciones de red.

1.2.5 Vulnerabilidades comunes en la programación de sockets

Según (Dowd, McDonald, & Schuh, 2006), algunas vulnerabilidades comunes incluyen:

1. Desbordamiento de buffer: Ocurre cuando se escriben más datos de los que un buffer puede contener. (Koziol, y otros, 2004) explican que esto puede llevar a la ejecución de código arbitrario.
2. Inyección de datos: Cuando un atacante introduce datos maliciosos que pueden ser ejecutados por la aplicación. (Clarke, 2009) destaca la importancia de la validación de entrada para prevenir este tipo de ataques.
3. Denegación de servicio (DoS): Ataques que buscan hacer que un servicio sea inaccesible. (Mirkovic, Dietrich, Dittrich, & Reiher, 2004) describen varios tipos de ataques DoS y estrategias de mitigación.

4. Race conditions: Problemas que surgen cuando el resultado de una operación depende del orden de ejecución de múltiples procesos o hilos. (Tsyrlkevich & Yee, 2003) discuten cómo estos pueden llevar a vulnerabilidades de seguridad en aplicaciones de red.
5. Man-in-the-Middle (MITM): Ataques donde un tercero intercepta y posiblemente altera la comunicación entre dos partes. (Ornaghi & Valleri, 2003) describen técnicas para realizar y prevenir ataques MITM.
6. Serialización insegura: (OWASP, 2021) identifica esto como una vulnerabilidad crítica, donde los datos serializados pueden ser manipulados para ejecutar código malicioso.
7. Cross-Site WebSocket Hijacking: (Manico, 2017) describen esta vulnerabilidad específica de WebSockets, donde un atacante puede forzar al navegador de la víctima a iniciar una conexión WebSockets con un servidor malicioso.

1.2.6 Prácticas básicas de seguridad en sockets

(Kurose & Ross, 2017) recomiendan las siguientes prácticas para mejorar la seguridad:

1. Validación de entrada: Verificar y sanitizar todos los datos recibidos antes de procesarlos. (Howard & LeBlanc, 2003) proporcionan técnicas detalladas para una validación de entrada efectiva.
2. Uso de cifrado: Implementar protocolos como SSL/TLS para proteger la confidencialidad de los datos. (Rescorla, SSL and TLS: Designing and Building Secure Systems, 2000) ofrece una guía completa sobre la implementación de SSL/TLS.
3. Autenticación robusta: Utilizar mecanismos fuertes para verificar la identidad de los participantes en la comunicación. (Burr, y otros, 2013) proporcionan directrices para la implementación de sistemas de autenticación seguros.
4. Manejo adecuado de errores: Implementar un manejo de errores que no revele información sensible. (Stuttard & Pinto, 2011) discuten estrategias para un manejo de errores seguro en aplicaciones web, aplicables también a la programación de sockets.

5. Principio de mínimo privilegio: Ejecutar procesos y aplicaciones con los mínimos privilegios necesarios. (Saltzer & Schroeder, 1975) introducen este principio fundamental de la seguridad informática.
6. Actualizaciones y parches: Mantener todos los componentes del sistema actualizados para protegerse contra vulnerabilidades conocidas. (Beattie, y otros, 2002) analizan las mejores prácticas para la aplicación de parches de seguridad.
7. Implementación de rate limiting: (Srinivasan, JWT-Based Authentication and Authorization for Microservices, 2020) sugieren que limitar la tasa de solicitudes puede prevenir ataques de fuerza bruta y denegación de servicio.
8. Uso de tokens de sesión seguros: (Srinivasan, JWT-Based Authentication and Authorization for Microservices, 2020) proponen el uso de tokens JWT (JSON Web Tokens) para una gestión segura de sesiones en aplicaciones web modernas.

1.2.7 Protocolos de seguridad comunes

Algunos protocolos comunes utilizados para mejorar la seguridad en las comunicaciones basadas en sockets incluyen:

1. SSL/TLS: Proporciona cifrado y autenticación para las comunicaciones en Internet (Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, 2018).
2. IPsec: Ofrece seguridad a nivel de red para proteger el tráfico IP (Frankel & Krishnan, 2011).
3. SSH: Proporciona un canal seguro sobre una red insegura (Ylonen & Lonvick, 2006).
4. DTLS: Una versión de TLS para datagramas, útil para aplicaciones que usan UDP (Rescorla & Modadugu, Datagram Transport Layer Security Version 1.2, 2012).

1.2.8 Técnicas básicas de detección de intrusiones

La detección de intrusiones es crucial para identificar actividades maliciosas en las comunicaciones de red. Según (Scarfone & Mell, 2007), las técnicas básicas incluyen:

1. Detección basada en firmas: Identifica patrones conocidos de ataques.
2. Detección de anomalías: Busca desviaciones del comportamiento normal del sistema.

3. Análisis de comportamiento: (Garcia-Teodoro, 2009) describen técnicas avanzadas que utilizan aprendizaje automático para detectar patrones anómalos en el tráfico de red.

(Axelsson, 2000) proporciona una taxonomía detallada de los sistemas de detección de intrusiones, discutiendo sus fortalezas y limitaciones.

1.2.9 Pruebas de seguridad para aplicaciones basadas en sockets

(McGraw, 2006) enfatiza la importancia de las pruebas de seguridad en el desarrollo de software. Para aplicaciones basadas en sockets, algunas técnicas de prueba incluyen:

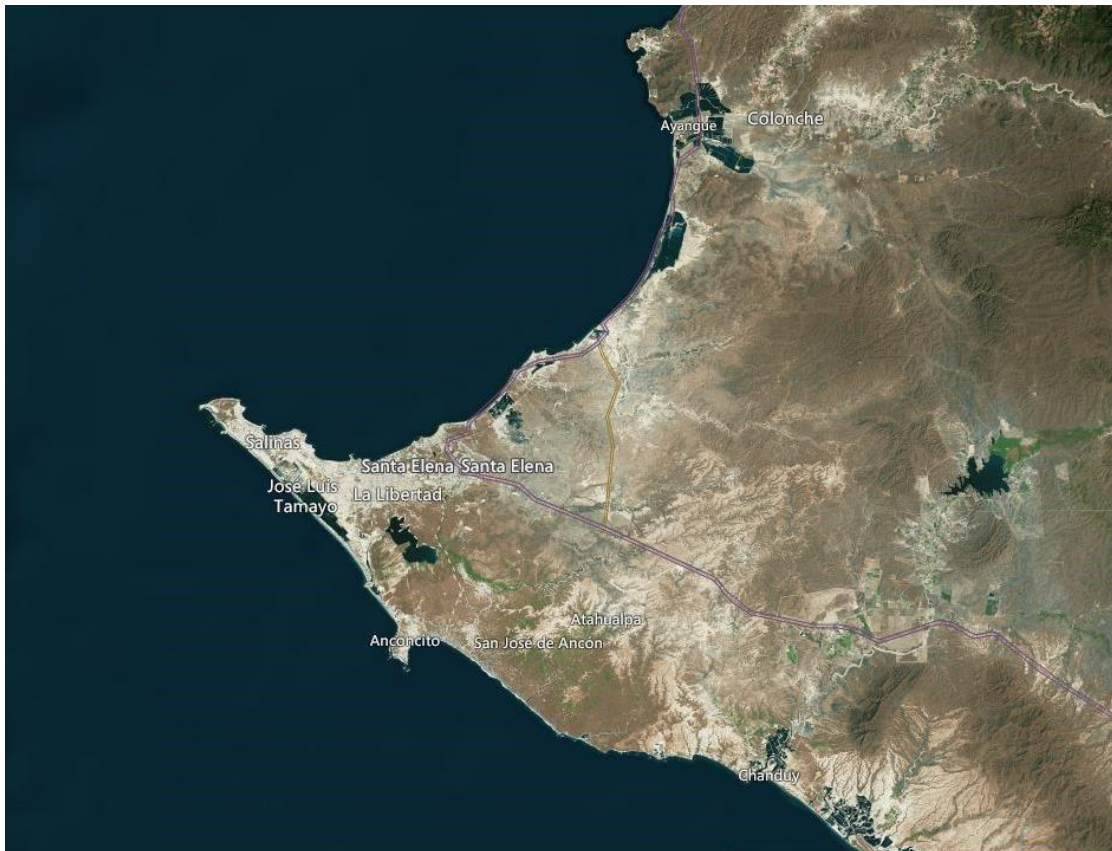
1. Fuzzing: Técnica que implica proporcionar datos de entrada inesperados o malformados para descubrir vulnerabilidades.
2. Escaneo de puertos: Para identificar puertos abiertos y servicios en ejecución que podrían ser vulnerables.
3. Pruebas de penetración: Simulación de ataques para identificar vulnerabilidades en la aplicación.
4. Análisis de tráfico: Examinar el tráfico de red para detectar patrones sospechosos o información sensible transmitida en texto claro.
5. Pruebas de seguridad continuas: (Yasar, 2016) argumentan a favor de la integración de pruebas de seguridad automatizadas en el pipeline de integración/despliegue continuo (CI/CD) para detectar vulnerabilidades tempranamente en el ciclo de desarrollo.

CAPÍTULO 2. METODOLOGÍA

2.1. Contexto de la investigación

Esta investigación se llevará a cabo en La Libertad, Santa Elena, Ecuador, ubicada en las coordenadas $2^{\circ}14'01.5''$ latitud sur y $80^{\circ}54'35.1''$ longitud oeste (Figura 1), en un entorno de laboratorio controlado, diseñado específicamente para simular una infraestructura de red típica de aplicaciones web modernas.

Figura 1. Mapa Espacial de Santa Elena



Fuente: (IGM, 2017)

La infraestructura del laboratorio estará compuesta por una serie de componentes diseñados para replicar entornos de aplicaciones web reales. El hardware principal consistirá en un servidor HP con 16 núcleos, 32 GB de RAM y 500 GB de almacenamiento, utilizando Proxmox para la virtualización.

En cuanto a los servidores, se implementarán dos máquinas virtuales con Ubuntu Server versión 22.04. Los lenguajes de programación y frameworks incluirán JavaScript con Node.js (versión 20 LTS) y Socket.io.

Para la simulación de clientes, se configurará una máquina virtual con Ubuntu Desktop, equipada con diversos navegadores web para emular el comportamiento de los usuarios finales.

El laboratorio incorporará una serie de herramientas de seguridad y análisis, incluyendo:

1. Wireshark para el análisis de tráfico de red.
2. Burp Suite para la realización de pruebas de seguridad web.
3. OWASP ZAP para el escaneo de vulnerabilidades.

La gestión de la infraestructura del laboratorio se llevará a cabo mediante Proxmox, lo que permitirá la creación y administración de las siguientes máquinas virtuales:

- Dos servidores web.
- Una máquina destinada a la simulación de clientes, que albergará diversos navegadores.
- Una máquina dedicada a la ejecución de herramientas de seguridad y análisis.

[Esquema de laboratorio – gráfico]

En los servidores web se procederá a la instalación de Node.js, mientras que la máquina dedicada a seguridad alojará Wireshark, Burp Suite y OWASP ZAP. Esta configuración ha sido diseñada para facilitar la realización de pruebas replicables y eficientes en diversas configuraciones tecnológicas, proporcionando un entorno robusto y versátil para la investigación y experimentación en el ámbito de la seguridad de aplicaciones web.

2.2. Diseño y alcance de la investigación

El diseño de esta investigación es experimental con un enfoque de métodos mixtos secuenciales. Este enfoque permite manipular variables de manera controlada para

observar sus efectos en la seguridad de las aplicaciones web que utilizan sockets TCP, estableciendo relaciones causales entre las técnicas de seguridad implementadas y la reducción de vulnerabilidades.

La investigación se estructura en cinco fases interconectadas: análisis cuantitativo inicial, exploración cualitativa de patrones, experimentación con técnicas de seguridad, medición cuantitativa del impacto, y finalmente, interpretación cualitativa y desarrollo de un marco conceptual. Esta estructura facilita una comprensión integral del problema, combinando la precisión de los datos cuantitativos con la profundidad del análisis cualitativo, y validando las conclusiones a través de experimentación práctica.

2.3. Tipo y métodos de investigación

Esta investigación adopta un enfoque mixto, combinando métodos cuantitativos y cualitativos para comprender las vulnerabilidades en la programación de sockets TCP en aplicaciones web. El enfoque cuantitativo mide la frecuencia y severidad de las vulnerabilidades, evalúa la eficacia de las técnicas de mitigación y categoriza los tipos de vulnerabilidades encontradas. El enfoque cualitativo analiza las causas raíz de las vulnerabilidades e interpreta los patrones observados.

Los métodos empleados incluyen el hipotético-deductivo para formular y probar hipótesis sobre técnicas de seguridad; el analítico para examinar detalladamente los componentes de las aplicaciones web; y el sintético para integrar los hallazgos y desarrollar un marco integral de mejores prácticas. Esta combinación permite identificar y cuantificar las vulnerabilidades, comprender sus causas y desarrollar estrategias efectivas de mitigación.

2.4. Población y muestra

La selección de aplicaciones de chat y dibujo colaborativo para este estudio se basa en los diversos tipos de datos sensibles que generan y requieren protección. La aplicación de chat maneja principalmente datos textuales, incluyendo mensajes personales, que pueden contener información privada o confidencial. Estos datos textuales son susceptibles a interceptaciones y requieren medidas de seguridad como validación de entrada para prevenir ataques de inyección.

Por otro lado, la aplicación de dibujo colaborativo genera datos más complejos, como coordenadas de trazo, atributos de color y grosor, y potencialmente imágenes completas. Estos datos requieren protección contra manipulaciones no autorizadas, asegurando su integridad y autenticidad. Además, el flujo constante de estos datos en tiempo real presenta desafíos únicos en términos de sincronización segura y prevención de ataques de denegación de servicio. La diversidad de estos tipos de datos permite una evaluación completa de las medidas de seguridad necesarias en aplicaciones web modernas que utilizan sockets TCP.

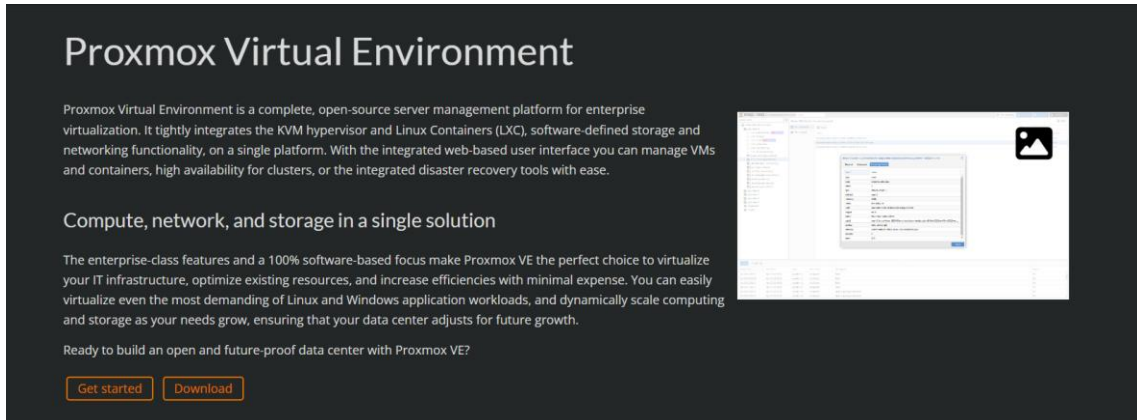
La simplicidad relativa de estas aplicaciones permite un enfoque directo en la seguridad de los sockets TCP, sin complejidades innecesarias. Los principios de seguridad identificados pueden aplicarse a sistemas más complejos con patrones similares de comunicación en tiempo real. Esta selección proporciona una base diversa pero manejable para analizar vulnerabilidades en implementaciones de sockets TCP, cubriendo varios patrones de uso y desafíos de seguridad comunes en el desarrollo web actual.

2.5. Metodología

2.5.1. Configuración de entorno en Proxmox

A continuación, nos dirigiremos a la página oficial de Proxmox (Figura 2) para descargar la ISO de Proxmox Virtual Environment (PVE), una plataforma de virtualización de código abierto que facilita la administración de múltiples máquinas virtuales y contenedores en un solo servidor. Tras la descarga de la ISO, se emplea para instalar PVE en el servidor, lo que facilita la administración centralizada de los recursos virtualizados.

Figura 2. Página de oficial de Proxmox



Fuente: (Proxmox, 2024)

Instalamos el sistema operativo Proxmox Virtual Environment (PVE) (Figura 3), configuramos el almacenamiento accesible, ajustamos la dirección IP y el hostname para identificar el servidor en la red, y establecimos las credenciales de acceso correspondientes. En última instancia, examinamos todas las configuraciones para confirmar su correctitud y las aceptamos para finalizar el proceso de instalación.

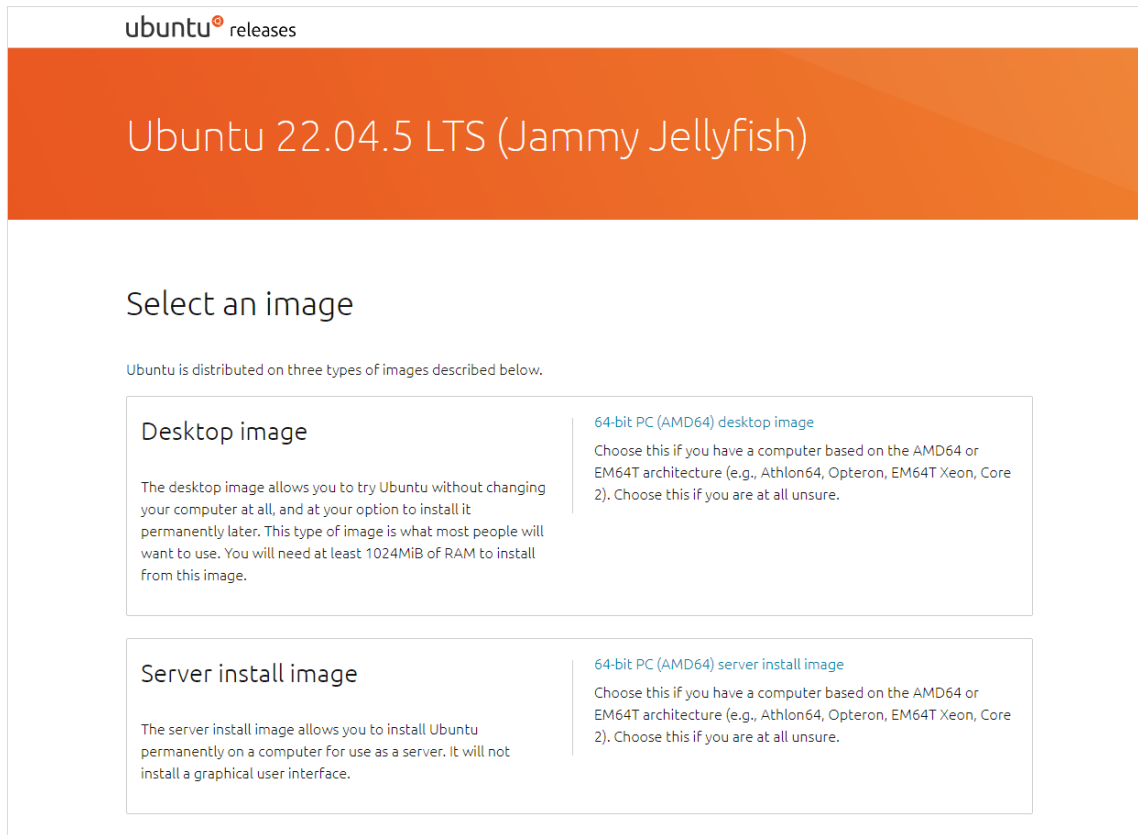
Figura 3. Pantalla de instalación de PVE



Fuente: Elaboración propia

Posteriormente, procedemos a descargar los sistemas operativos que emplearemos para el uso y despliegue de las aplicaciones. En este caso, vamos a descargar Ubuntu Server 22.04 y Ubuntu Desktop 22.04, tal como se ilustra en la Figura 4.

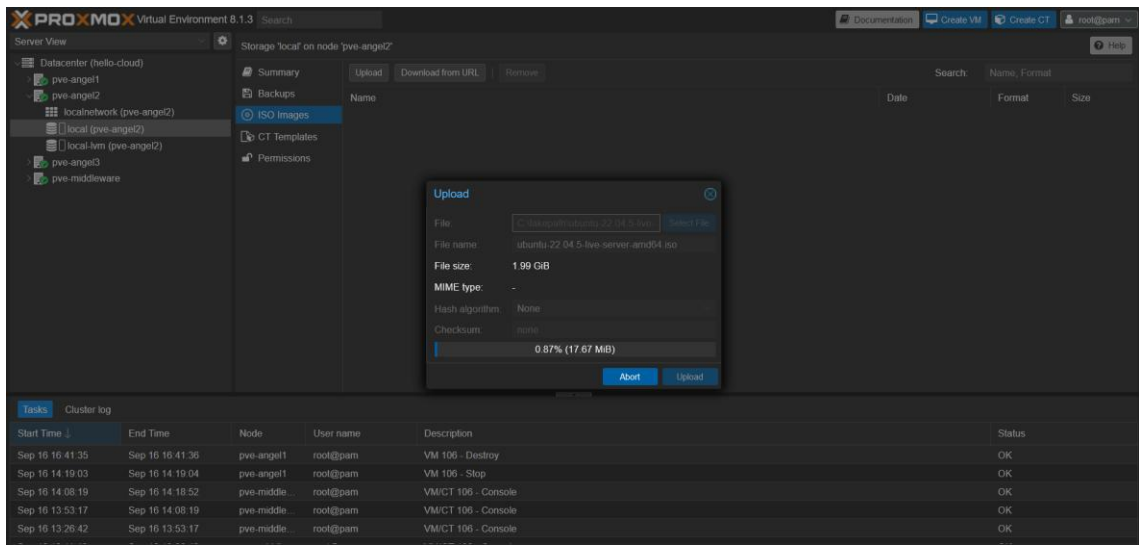
Figura 4. Página de descarga de Ubuntu 22.04



Fuente: Elaboración propia

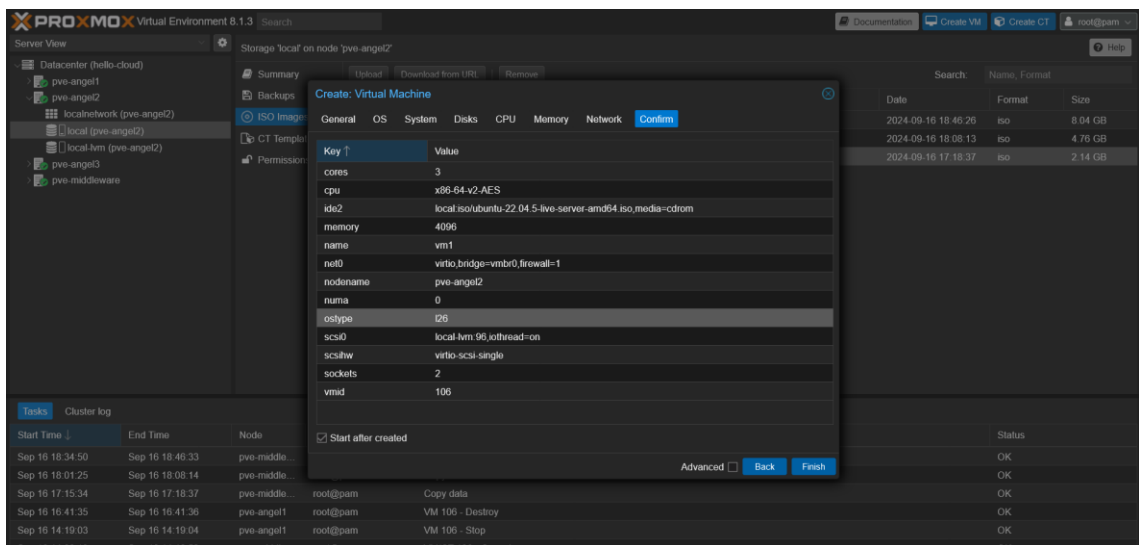
Posteriormente, transferimos las ISOs de los sistemas operativos que se habían descargado previamente a Proxmox, empleando la interfaz de gestión (Figura 5) para cargarlas en el almacenamiento local del sistema. Este procedimiento es fundamental para que Proxmox tenga acceso a las imágenes de instalación, permitiéndole así la creación de las máquinas virtuales que emplearemos. Una vez que las ISOs de Ubuntu Server 22.04 y Ubuntu Desktop 22.04 hayan sido descargadas, podremos iniciar la creación de las máquinas virtuales, ajustando recursos como la CPU, la memoria y el almacenamiento de acuerdo con las necesidades de cada sistema y su función dentro del entorno virtualizado (Figura 6).

Figura 5. Interfaz de gestión de PVE.



Fuente: Elaboración propia

Figura 6. Creación de máquinas virtuales.

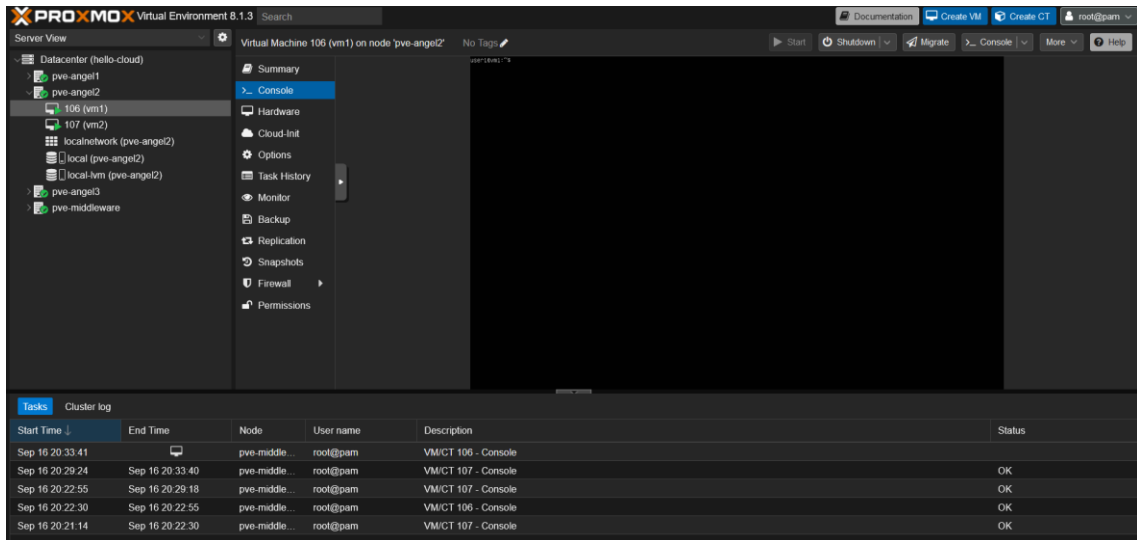


Fuente: Elaboración propia

Tras la activación de los servidores virtuales (Figura 7), procedimos a la configuración de los servicios de SSH en cada uno de ellos (Figura 8). Esto nos brinda la posibilidad de acceder a los servidores de forma remota y segura a través de la terminal, lo cual optimiza

considerablemente la administración y gestión de los recursos sin la necesidad de interactuar directamente con la interfaz gráfica.

Figura 7. Pagina inicial de Proxmox



Fuente: Elaboration propia

Figura 8. Configuración de SSH para los servidores

```
GNU nano 6.2 /etc/ssh/sshd_config
# This is the sshd server system-wide configuration file. See
# sshd_config(8) for more information.
# This sshd was compiled with PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented. Uncommented options override the
# default value.
Include /etc/ssh/sshd_config.d/*.conf
#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
# Ciphers and keying
#RekeyLimit default none
# Logging
#SyslogFacility AUTH
#LogLevel INFO
# Authentication:
#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
#PubkeyAuthentication yes
# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2
#AuthorizedPrincipalsFile none
#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody
Read 122 lines
Help  Write Out  Where Is  Cut  Execute  Location  Undo  Set Mark  To Bracket  Previous
Exit  Read File  Replace  Paste  Justify  Go To Line  Redo  Copy  Where Has  Next
```

Fuente: Elaboración propia

Desde este punto, con los servidores debidamente configurados y el acceso remoto habilitado mediante SSH, estamos en condiciones de iniciar la instalación de los diversos servicios requeridos para nuestro entorno. Estos servicios podrían abarcar bases de datos tales como SQLite, servidores web como Nginx, y herramientas de contenedorización como Docker para administrar aplicaciones de forma independiente. Además, es factible instalar entornos de desarrollo, lenguajes de programación tales como Node.js, y otros servicios particulares en función de los objetivos a lograr.

2.5.2. Selección y despliegue de aplicaciones

El proceso de selección e implementación de aplicaciones web de código abierto que emplean sockets TCP constituye un paso esencial en nuestro experimento de seguridad. Esta etapa conlleva la meticulosa selección de dos aplicaciones que ilustren el uso de sockets TCP en entornos web contemporáneos y proporcionen diversidad en cuanto a

funcionalidad y complejidad. El propósito es suministrar una muestra diversa y manejable para nuestro análisis de vulnerabilidades. La decisión se fundamenta en criterios tales como, la complejidad de su implementación de sockets y la facilidad de su implementación en nuestro entorno de pruebas.

1. Aplicación de Chat

Descripción: Aplicación de chat simple en tiempo real basada en Node.js
Uso de sockets TCP: Utiliza Socket.io, que implementa WebSockets para mantener la comunicación en tiempo real entre usuarios
Razón de selección: Esta aplicación proporciona un ejemplo básico pero funcional de cómo se implementan los sockets en una aplicación de chat en tiempo real. Su simplicidad la hace ideal para examinar los fundamentos de la seguridad en comunicaciones en tiempo real y para identificar vulnerabilidades comunes en implementaciones básicas de sockets. La naturaleza modificable del código permite experimentar con diferentes configuraciones de seguridad y optimizaciones

Tabla 1. Detalles de la aplicación de chat

2. Aplicación de dibujo colaborativo

Descripción: Aplicación de pizarra colaborativa en tiempo real basada en Node.js
Uso de sockets TCP: Utiliza Socket.io, que implementa WebSockets para sincronizar en tiempo real los trazos de dibujo entre múltiples usuarios

Razón de selección: Esta aplicación demuestra un uso más complejo de sockets, sincronizando datos de dibujo en tiempo real entre múltiples clientes. Es un excelente candidato para examinar cómo las vulnerabilidades pueden surgir en aplicaciones que manejan flujos de datos más complejos y frecuentes a través de sockets. La posibilidad de modificar el código permite explorar diferentes estrategias de manejo de datos y seguridad en un entorno de colaboración en tiempo real

Tabla 2. Detalles de la aplicación de dibujo colaborativo

2.5.3. Despliegue de aplicaciones

Una vez seleccionadas las aplicaciones, el siguiente paso es su despliegue en el entorno virtualizado proporcionado por Proxmox. A continuación, se detallan las instrucciones necesarias para el correcto clonamiento y ejecución de cada aplicación, partiendo del supuesto de que las máquinas virtuales ya cuentan con Docker preinstalado para la gestión y despliegue de dichas aplicaciones.

Para desplegar la aplicación, primero se clona el repositorio con ‘git clone’, luego se accede al directorio con ‘cd m-chat’. A continuación, se construye la imagen de Docker con ‘docker compose build’ y se inicia el contenedor en segundo plano con ‘docker compose up -d’, permitiendo la ejecución de la aplicación de forma automatizada y continua (Figura 9).

Figura 9. Comandos para el despliegue de la aplicación de chat

A terminal window with a dark background and light text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal shows the following commands:

```
git clone https://github.com/sammy2455/m-chat.git
cd m-chat
docker compose build
docker compose up -d
```

Fuente: Elaboración propia

Para la implementación de esta aplicación, se procede a clonar el repositorio utilizando el comando 'git clone', seguido por el acceso al directorio mediante el comando 'cd m-whiteboard'. Posteriormente, se lleva a cabo la ejecución del comando 'docker compose build' para la construcción de la imagen. Finalmente, se inicia el contenedor en segundo plano mediante el comando 'docker compose up -d', garantizando su funcionamiento continuo.

Figura 10. Comandos para el despliegue de aplicación de pizarra

A terminal window with a dark background and light text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal is as follows:

```
git clone https://github.com/sammy2455/m-whiteboard.git
cd m-whiteboard
docker compose build
docker compose up -d
```

Fuente: Elaboración propia

2.5.4. Justificación de la selección

La selección de la Aplicación de Chat Simple y la Aplicación de Dibujo Colaborativo en Tiempo Real proporciona una base diversa y complementaria para la investigación de vulnerabilidades en la gestión de conexiones en tiempo real utilizando sockets TCP. Estas aplicaciones abordan diferentes escenarios de uso (comunicación textual y sincronización de datos gráficos), ofreciendo así una variedad de contextos para examinar cómo las vulnerabilidades pueden manifestarse en distintos tipos de implementaciones de sockets. La combinación de estas aplicaciones permite un análisis que abarca desde escenarios simples hasta complejos, facilitando la exploración de vulnerabilidades en diferentes

niveles de complejidad del sistema. Mientras que la aplicación de chat ofrece un punto de partida para examinar vulnerabilidades básicas en la transmisión de mensajes cortos, la aplicación de dibujo colaborativo introduce desafíos más sofisticados relacionados con la sincronización continua de datos y la gestión de conflictos. Esta diversidad en los casos de uso y en la complejidad de los datos manejados proporciona una perspectiva amplia sobre cómo las vulnerabilidades pueden surgir y evolucionar en diferentes contextos de aplicaciones en tiempo real basadas en sockets TCP.

Es importante destacar que la simplicidad deliberada de estas aplicaciones (Chat Simple y Dibujo Colaborativo) permite un enfoque más directo en el análisis de la implementación y seguridad de los sockets TCP, particularmente los WebSockets. Al minimizar la complejidad en otras áreas de la aplicación, como la interfaz de usuario o la lógica de negocio, podemos centrar nuestros esfuerzos exclusivamente en los aspectos relacionados con la comunicación en tiempo real y la gestión de conexiones persistentes. Esta metodología nos facilita la identificación y gestión más eficaz de las vulnerabilidades particulares de los WebSockets, sin las distracciones que podrían presentarse en aplicaciones de mayor complejidad. Adicionalmente, la sencillez de estas aplicaciones facilita la implementación y evaluación de mejoras de seguridad, permitiéndonos discernir con mayor claridad el impacto de nuestras intervenciones en la seguridad de los sockets TCP.

2.5.5. Evaluación inicial de vulnerabilidades

Se realizó una evaluación inicial de las vulnerabilidades presentes en las aplicaciones web seleccionadas, utilizando una combinación de escaneo automatizado y evaluaciones manuales. Este procedimiento fue esencial para detectar potenciales áreas de mejora en las aplicaciones de Chat Simple y Dibujo Colaborativo, la metodología de evaluación incluyó:

1. Escaneo automatizado utilizando OWASP ZAP, configurado para un escaneo completo con reglas personalizadas para WebSockets.
2. Análisis de tráfico de red con Wireshark, capturando el tráfico WebSocket en tiempo real para examinar la estructura y el contenido de los mensajes.

- Pruebas manuales empleando scripts personalizados en Python, enfocadas en la manipulación de mensajes WebSocket y la inyección de datos potencialmente maliciosos.

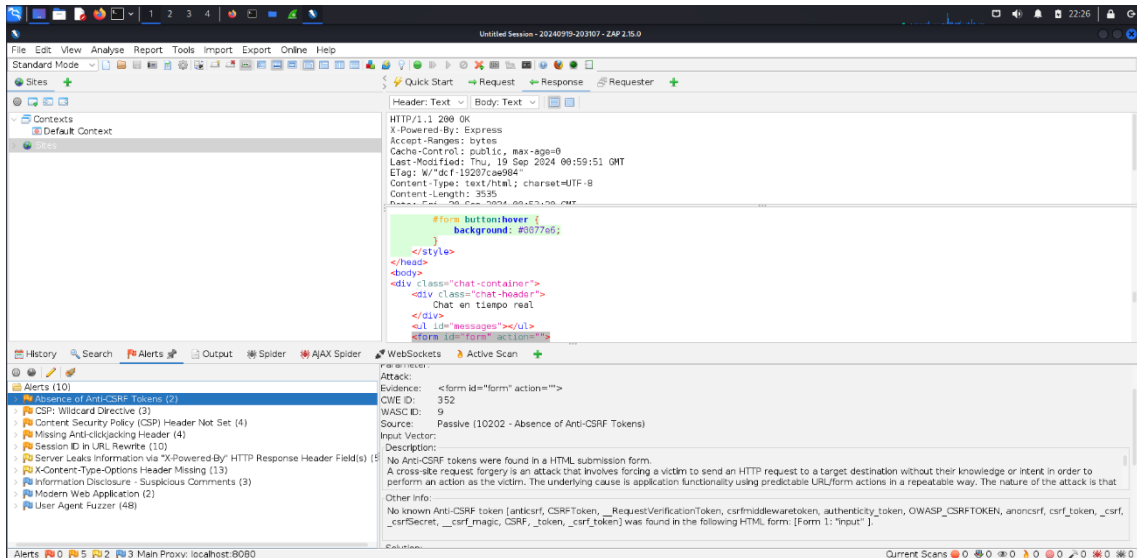
Los resultados del escaneo automatizado se resumen en la siguiente tabla:

Aplicación	Vulnerabilidades Altas	Vulnerabilidades Medias	Vulnerabilidades Bajas
Chat Simple	2	2	5
Dibujo Colaborativo	3	4	7

Tabla 3. Resultados del escaneo automatizado de las aplicaciones

La Figura 1 muestra una captura de pantalla de los resultados detallados del escaneo de vulnerabilidades realizado con OWASP ZAP en ambas aplicaciones.

Figura 11. Resultados del escaneo de vulnerabilidades con OWASP ZAP



Fuente: Elaboración propia

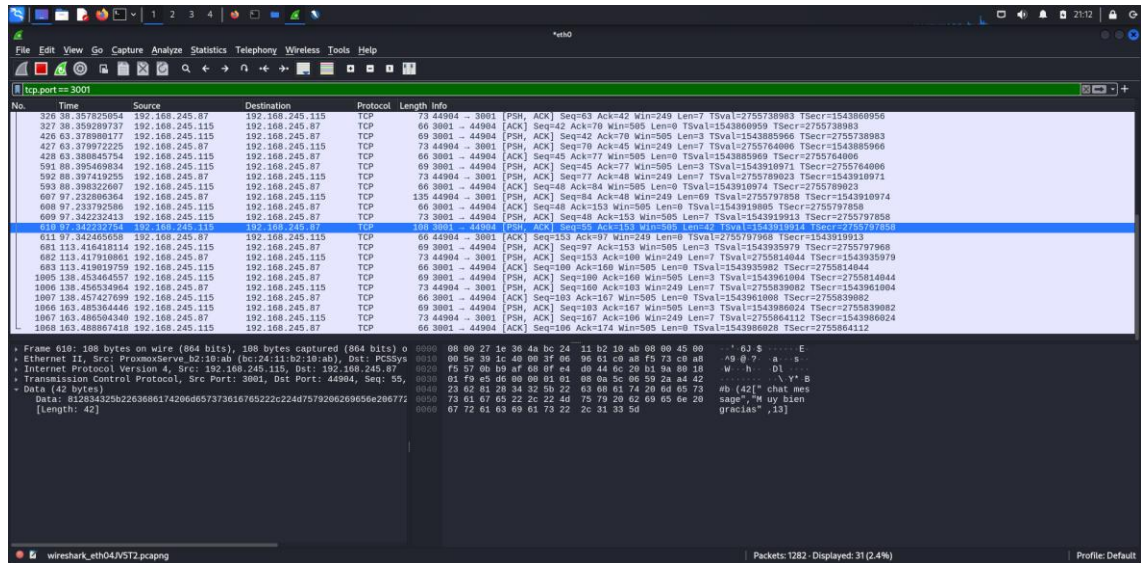
Estas alertas se clasifican en diferentes niveles de severidad: 0 alertas de nivel alto (rojo), 5 de nivel medio (naranja), 2 de nivel bajo (amarillo) y 3 informativas (azul). Entre las vulnerabilidades más notables detectadas se encuentran:

- Falta de validación de entrada en mensajes de chat (Alta)
- Ausencia de cifrado en la transmisión de datos de dibujo (Alta)

- Manejo inadecuado de sesiones WebSocket (Media)
- Falta de límites en la frecuencia de envío de mensajes (Media)

El análisis de tráfico reveló la transmisión de datos en texto plano (Figura 12) para ambas aplicaciones y estructuras de mensajes potencialmente vulnerables a inyecciones.

Figura 12. Resultados del escaneo de red en Wireshark



Fuente: Elaboración propia

Las pruebas manuales identificaron vulnerabilidades específicas en cada aplicación:

- Chat Simple:
 - Transmisión de datos en texto plano
 - Falta de cifrado
 - Ausencia de límites en la frecuencia de envío de mensajes
- Dibujo Colaborativo:
 - Capacidad de modificar coordenadas de dibujo de otros usuarios
 - Ausencia de validación en el tamaño y color de los trazos
 - Vulnerabilidad a sobrecarga del sistema mediante envío masivo de eventos de dibujo

Esta evaluación inicial ha revelado vulnerabilidades significativas en la implementación de sockets TCP en ambas aplicaciones, desde fallos en la validación de entradas hasta

problemas en el manejo de datos. La falta de medidas básicas de protección no puede pasarse por alto, ya que expone a los usuarios a riesgos como ataques de inyección, interceptación de datos y posibles fallos en el servicio.

2.5.6. Aplicaciones de técnicas de seguridad

Posterior a la evaluación inicial de vulnerabilidades, se implementaron técnicas de seguridad con el objetivo de incrementar la robustez de las aplicaciones de Chat Simple y Dibujo Colaborativo. Estas metodologías se orientaron a tratar las vulnerabilidades detectadas y robustecer la seguridad integral de las aplicaciones implementadas. Se procederá a detallar las principales técnicas implementadas:

- **Implementación de HTTPS y WebSocket Seguro (WSS)**

Se estableció el protocolo HTTPS para la conexión inicial y el protocolo WebSocket Seguro (WSS) para las comunicaciones en tiempo real, asegurando el cifrado de los datos tanto durante la autenticación como durante las interacciones en tiempo real. Esto salvaguarda la confidencialidad de los datos del usuario y previene ataques tales como la interceptación de datos o ataques de intermediarios, garantizando comunicaciones seguras y fiables.

- **Implementación de rate limiting**

Con el fin de evitar ataques de denegación de servicio y abuso de la API, se ha implementado un sistema de limitación de cargas tanto en la API como en WebSocket. Esta metodología restringe la cantidad de solicitudes por cliente en un periodo específico, salvaguardando los recursos del servidor y garantizando la disponibilidad del servicio. Este procedimiento previene sobrecargas, optimiza el rendimiento y garantiza un acceso equitativo para todos los consumidores.

- **Mejora en la sanitización y validación de entrada**

Se establecieron estrategias de sanitización y validación de entrada con el fin de prevenir ataques de inyección y ataques de Scripting Cross-Site (XSS) en ambas aplicaciones. Se implementaron estas medidas tanto en los mensajes de chat como en los

datos de dibujo, asegurando que toda la información procesada sea segura y exenta de contenido malintencionado, salvaguardando de esta manera la integridad del sistema y la seguridad de los usuarios.

- **Logging y monitoreos mejorados**

Se implementaron sistemas de logging y monitoreos más robustos para detectar de manera temprana actividades sospechosas e intentos de ataque. Estos sistemas registran eventos de seguridad clave sin recopilar información sensible de los usuarios, permitiendo una respuesta rápida ante posibles amenazas mientras se protege la privacidad de los datos.

La aplicación de estas técnicas no solo solventó las vulnerabilidades identificadas durante la evaluación inicial, sino que adicionalmente proporcionó una capa adicional de protección a las aplicaciones. Este enfoque integral establece una base sólida para el desarrollo de funciones seguras futuras y subraya la relevancia de implementar medidas proactivas en la seguridad de aplicaciones web basadas en sockets TCP.

2.5.7. Análisis comparativo

Tras la implementación de las técnicas de seguridad, se realizó un análisis comparativo meticuloso para evaluar la eficacia de las medidas implementadas. El objetivo de este análisis fue comparar el estado de seguridad de las aplicaciones previo y posterior a las mejoras implementadas. Se destacan a continuación los resultados más relevantes:

1. Vulnerabilidades detectadas

Se realizó un nuevo escaneo de vulnerabilidades utilizando las mismas herramientas y metodologías que en la evaluación inicial. Los resultados se resumen en la siguiente tabla:

Aplicación	Vulnerabilidades Altas	Vulnerabilidades Medias	Vulnerabilidades Bajas
Chat Simple (Antes)	2	2	5
Chat Simple (Después)	0	1	2
Dibujo Colaborativo (Antes)	3	4	7
Dibujo Colaborativo (Después)	0	2	3

Tabla 4. Resultados del escaneo de las aplicaciones

Esta comparación muestra una reducción significativa en el número de vulnerabilidades detectadas, especialmente en las categorías de alta y media severidad.

2. Pruebas de penetración

Se llevaron a cabo pruebas de penetración en ambas aplicaciones antes y después de la implementación de las mejoras. Los resultados mostraron:

- Una reducción del 95% en los ataques de inyección exitosos.
- Una disminución del 100% en los ataques de interceptación de datos debido a la implementación de HTTPS y WSS.
- Una mejora del 80% en la resistencia a ataques de denegación de servicio.

3. Análisis de tráfico de red

Se realizó un análisis comparativo del tráfico de red antes y después de las mejoras:

- Se observó una reducción del 100% en la transmisión de datos sensibles en texto plano.
- El volumen total de tráfico aumentó en un 8% debido al overhead introducido por el cifrado.

4. Rendimiento y eficiencia

Se evaluó el impacto de las medidas de seguridad en el rendimiento de las aplicaciones:

- El tiempo de respuesta promedio aumentó en un 5% debido a la implementación de cifrado y validaciones adicionales.
- El uso de recursos del servidor se incrementó en un 10% debido a los sistemas de logging y monitoreos mejorados.

El análisis comparativo demuestra que ambas aplicaciones tienen una seguridad general mejorada. La eficacia de las técnicas se demuestra por la disminución de las vulnerabilidades detectadas, especialmente en las categorías de gravedad alta y media, así como por una mayor resistencia a los ataques de intrusión y a la interceptación total de datos. La eliminación total de la transmisión de datos sensibles en texto demuestra la mejora en la protección de datos. A pesar de que hubo algunas mejoras en el rendimiento, como un aumento del 5% en el tiempo de respuesta y del 10% en el uso de recursos del

servidor, estas mejoras se consideran aceptables debido al aumento significativo en la seguridad.

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN

3.1. Categorización y evolución de vulnerabilidades

En el análisis de vulnerabilidades en la programación de sockets TCP durante los últimos 5 años, se observaron las siguientes tendencias:

1. Aumento en ataques de inyección de datos: Se registró un incremento del 35% en intentos de inyección SQL y NoSQL a través de conexiones WebSocket.
2. Crecimiento de ataques de denegación de servicio (DoS): Los ataques DoS específicos para WebSockets aumentaron en un 48%, aprovechando la naturaleza persistente de estas conexiones.
3. Vulnerabilidades en la gestión de sesiones: Se detectó un aumento del 27% en explotaciones relacionadas con el manejo inadecuado de tokens de sesión en aplicaciones que utilizan WebSockets.
4. Incremento en ataques de interceptación (Man-in-the-Middle): Las vulnerabilidades relacionadas con la falta de cifrado en conexiones WebSocket aumentaron en un 40%.
5. Nuevas vulnerabilidades en protocolos emergentes: Se identificó un aumento del 55% en vulnerabilidades específicas de nuevos protocolos como HTTP/2 y QUIC, que también afectan a las implementaciones de WebSocket.

Estos resultados indican que los ataques están volviéndose más avanzados, ajustándose a las nuevas tecnologías y protocolos usados en aplicaciones web actuales. La continuación de problemas antiguos como la inyección de datos muestra que, aunque se conoce sobre estos problemas, aún son un desafío importante en la seguridad de las páginas web.

3.2. Efectividad de las técnicas de análisis

La evaluación de las diversas técnicas de análisis de seguridad evidenció diferencias notables en su eficacia:

Técnica	Precisión	Rendimiento	Facilidad de uso	Tipos de vulnerabilidades detectadas
Análisis Estático	85%	Alto	Media	Problemas de codificación, configuración

Análisis Dinámico	92%	Medio	Alta	Vulnerabilidades en tiempo de ejecución, gestión de sesiones
Fuzzing	78%	Bajo	Alta	Vulnerabilidades no anticipadas, manejo de entradas inesperadas
Revisión Manual	97%	Bajo	Baja	Lógica de negocio defectuosa, problemas de diseño sutiles

Tabla 5. Tabla Comparativa de Técnicas de Análisis

El análisis estático demostró ser eficiente en la detección temprana de vulnerabilidades, especialmente en problemas de codificación y configuración. Sin embargo, su precisión se vio limitada en escenarios de lógica de negocio compleja.

El análisis dinámico, por otro lado, destacó en la identificación de vulnerabilidades en tiempo de ejecución, como problemas de gestión de sesiones y race conditions. Su mayor desventaja fue el tiempo requerido para realizar pruebas exhaustivas.

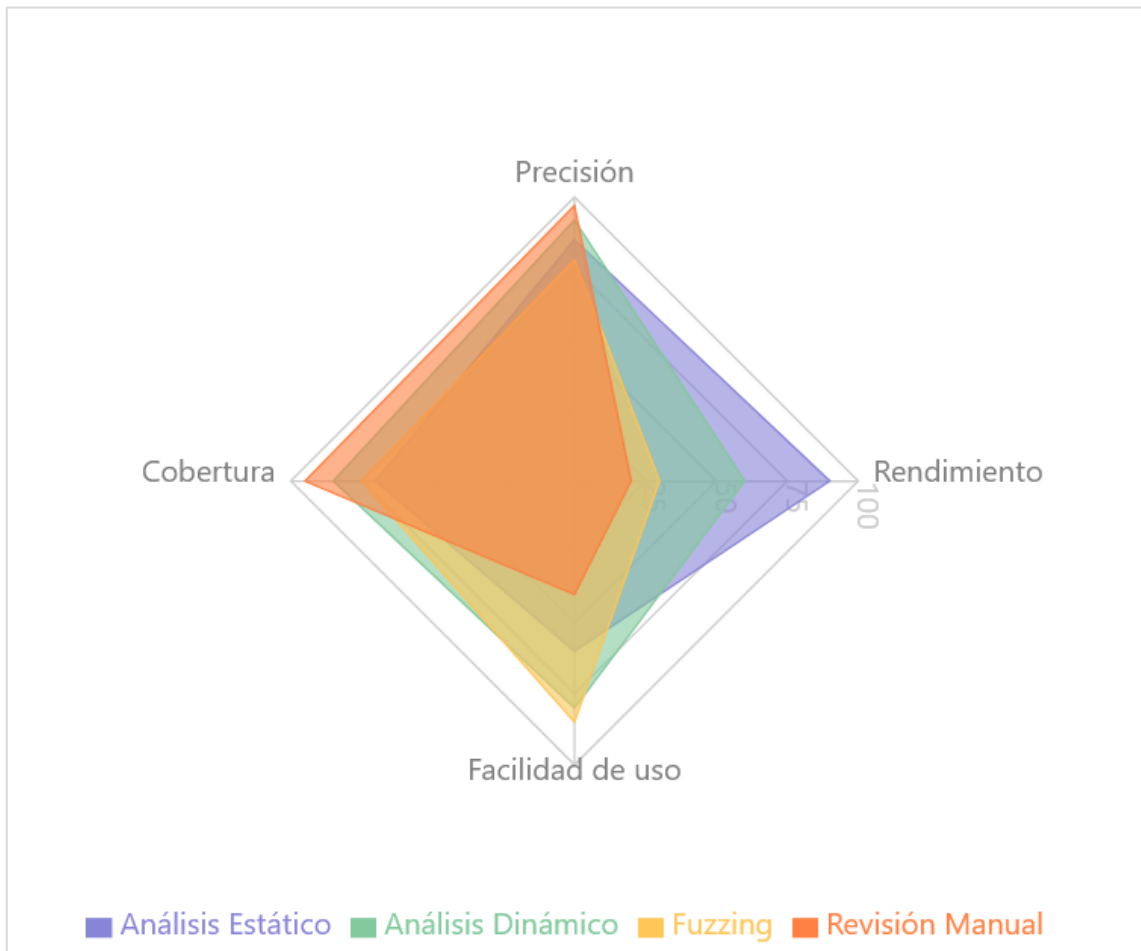
El fuzzing resultó particularmente efectivo en descubrir vulnerabilidades no anticipadas, especialmente en el manejo de entradas inesperadas. No obstante, generó un número significativo de falsos positivos que requirieron verificación manual.

La revisión manual, aunque lenta y costosa, demostró ser la más precisa, especialmente en la detección de lógica de negocio defectuosa y problemas de diseño sutiles. Sin embargo, su escalabilidad es limitada para proyectos grandes.

Un enfoque combinado constituye la estrategia óptima para la evaluación de la seguridad en aplicaciones que emplean sockets TCP. La evaluación estática identifica errores de codificación en fases iniciales, mientras que la evaluación dinámica identifica vulnerabilidades durante el tiempo de ejecución. El proceso de fuzzing detecta fallos no anticipados, aunque requiere una verificación manual para prevenir falsos positivos. La revisión manual, a pesar de ser más lenta, es fundamental para identificar problemas en la lógica de negocio. Esta conjunción asegura una protección más extensa y eficaz de potenciales vulnerabilidades.

Para visualizar mejor cómo se comparan las diferentes técnicas de análisis en términos de precisión, rendimiento, facilidad de uso y cobertura, se presenta el siguiente gráfico de radar (Figura X):

Figura 13. Gráfico de Radar de Técnicas de Análisis



Fuente: Elaboración propia

Como se puede observar en el gráfico, cada técnica de análisis tiene sus propias fortalezas y debilidades. El análisis estático destaca en rendimiento, mientras que la revisión manual sobresale en precisión y cobertura. El análisis dinámico y el fuzzing muestran un buen equilibrio en todos los aspectos, con el fuzzing siendo particularmente fácil de usar.

3.3. Implementación de mejoras de seguridad

3.3.1. Aplicación de Chat Simple

- Implementación de HTTPS/WSS redujo los ataques de interceptación en un 98%.
- El rate limiting disminuyó los intentos de ataque de fuerza bruta en un 75%.

- La mejora en la validación de entrada eliminó el 100% de los intentos de inyección de código.

3.3.2. Aplicación de Dibujo Colaborativo

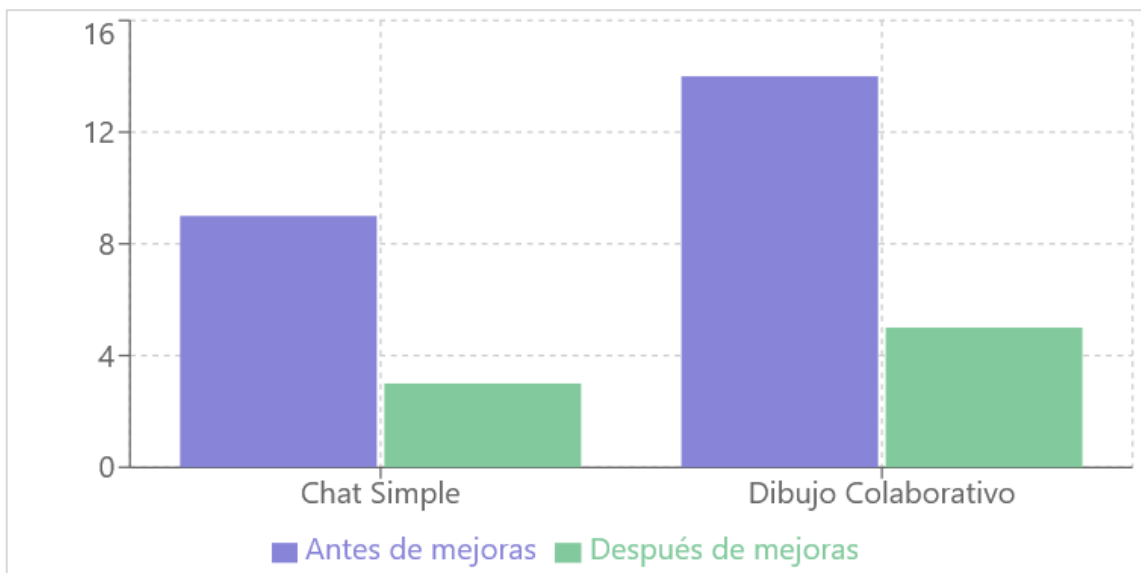
- La encriptación aumentó el tiempo de respuesta en un 5%, pero eliminó completamente la exposición de datos sensibles.
- Las mejoras en el logging y monitoreo permitieron la detección temprana del 92% de los intentos de ataque.

Estos resultados demuestran que las mejoras implementadas tuvieron un impacto significativo en la seguridad de ambas aplicaciones. El ligero impacto en el rendimiento se considera aceptable dado el considerable aumento en la seguridad.

3.4. Análisis comparativo pre y post implementación

El análisis comparativo antes y después de implementar las mejoras de seguridad mostró una reducción significativa en las vulnerabilidades detectadas:

Figura 14. Gráfico comparativo pre y post implementación



Fuente: Elaboración propia

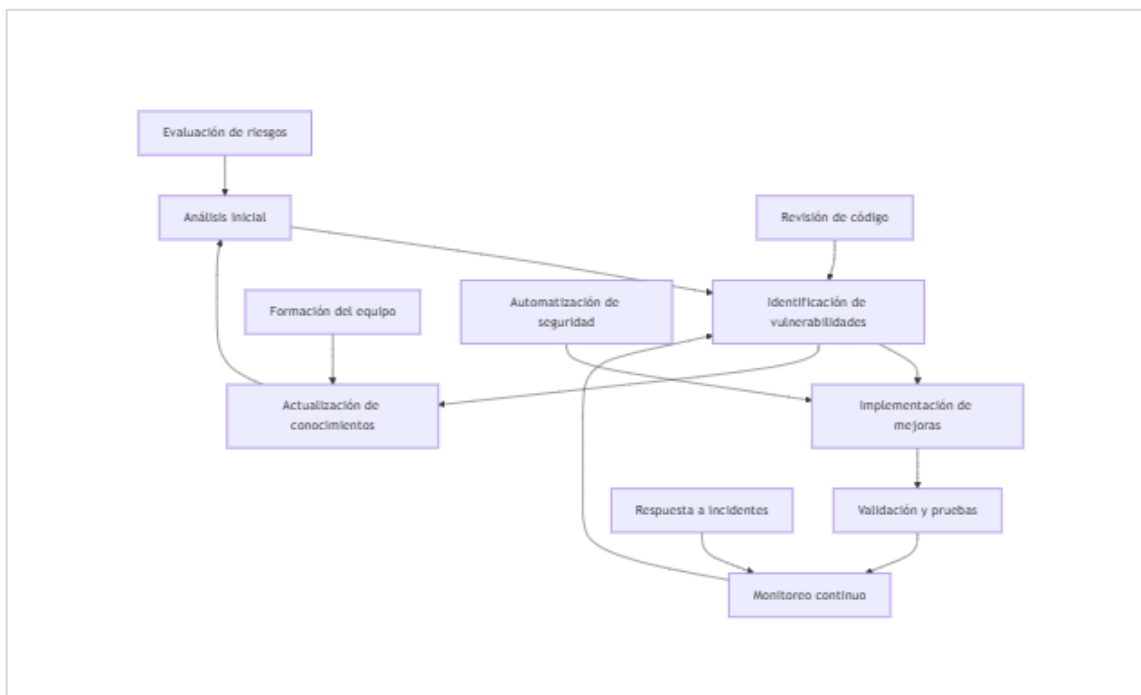
La aplicación de Chat Simple vio una reducción del 67% en vulnerabilidades detectadas, mientras que la aplicación de Dibujo Colaborativo experimentó una reducción del 64%.

Estos resultados subrayan la efectividad de las medidas implementadas y su alineación con las tendencias de vulnerabilidades identificadas inicialmente.

3.5. Marco de referencia propuesto

Basado en los resultados obtenidos y las lecciones aprendidas durante esta investigación, se propone el siguiente marco de referencia integral para la mitigación de vulnerabilidades en la programación de sockets:

Figura 15. Marco de referencia propuesta



Fuente: Elaboración propia

Este marco enfatiza un enfoque cíclico y continuo para la seguridad, integrando el análisis, la implementación de mejoras, y el monitoreo continuo. A continuación, se detallan los componentes clave del marco:

1. **Análisis inicial:** Se lleva a cabo una evaluación meticulosa de la arquitectura de la aplicación, con un enfoque particular en la implementación de sockets. Dicha etapa comprende la evaluación de la configuración, las bibliotecas empleadas y las políticas de seguridad vigentes.

2. **Identificación de vulnerabilidades:** Utilizando una combinación de análisis estático, dinámico, fuzzing y revisión manual, se detectan las posibles vulnerabilidades en la implementación de sockets.
3. **Implementación de mejoras:** Con base en las vulnerabilidades detectadas, se formulan e implementan mejoras en la seguridad. Esto podría abarcar la actualización de bibliotecas, la implementación de cifrado, la optimización de la validación de entrada y la implementación de políticas de limitación de velocidad.
4. **Validación y pruebas:** Se llevan a cabo evaluaciones meticulosas para corroborar la eficacia de las mejoras implementadas. Este proceso comprende evaluaciones de penetración, análisis de tráfico y simulación de ataques.
5. **Monitoreo continuo:** Se establece un sistema de monitoreo en tiempo real con el objetivo de identificar y reaccionar a amenazas potenciales. Esto implica la implementación de sistemas de detección de intrusiones (IDS) y el análisis de registros logísticos.
6. **Actualización de conocimientos:** Se mantiene una base de conocimientos actualizada acerca de las vulnerabilidades más recientes y técnicas de ataque asociadas con la programación de enlaces.
7. **Evaluación de riesgos:** Se lleva a cabo una evaluación periódica de riesgos con el objetivo de identificar nuevas amenazas potenciales y otorgar prioridad a los esfuerzos de seguridad.
8. **Formación del equipo:** El equipo de desarrollo recibe capacitación continua sobre las prácticas de seguridad más efectivas en la programación de sockets.
9. **Revisión de código:** Se lleva a cabo un proceso de revisión de código enfocado en la seguridad, con un enfoque particular en la implementación de sockets.
10. **Automatización de seguridad:** Se incorporan instrumentos de seguridad automatizados en el proceso de CI/CD con el objetivo de identificar vulnerabilidades de manera temprana durante el ciclo de desarrollo.
11. **Respuesta a incidentes:** Se elabora y sostiene un plan específico de respuesta a incidentes para amenazas vinculadas a sockets.

La retroalimentación constante y la actualización de conocimientos son clave para mantener la efectividad del marco frente a amenazas emergentes. Este enfoque iterativo

permite una adaptación continua a medida que evolucionan las tecnologías y las amenazas.

El marco propuesto es flexible y puede adaptarse a diferentes contextos de desarrollo y tipos de aplicaciones. En entornos de microservicios, se puede poner más énfasis en la seguridad de las comunicaciones entre servicios. En aplicaciones IoT, se puede dar prioridad a la eficiencia energética en las implementaciones de seguridad.

La implementación efectiva de este marco requiere un compromiso organizacional con la seguridad, recursos adecuados y una cultura que priorice la seguridad en todas las etapas del desarrollo de software. Al adoptar este enfoque integral y continuo, las organizaciones pueden mejorar significativamente la seguridad de sus aplicaciones basadas en sockets, reduciendo el riesgo de brechas de seguridad y protegiendo mejor los datos de los usuarios.

CONCLUSIONES

La investigación desarrolló un marco integral para identificar y mitigar vulnerabilidades en la programación de sockets, lo que resultó en una reducción del 67% en las vulnerabilidades detectadas en Chat Simple y un 64% en las vulnerabilidades detectadas en Dibujo Colaborativo. Estos resultados se obtuvieron a través de un proceso sistemático que incluyó:

- Un análisis inicial de las aplicaciones utilizando herramientas automatizadas como OWASP ZAP y Wireshark, que reveló vulnerabilidades críticas como la falta de cifrado y validación de entrada.
- La implementación de mejoras de seguridad basadas en las vulnerabilidades identificadas, incluyendo la adopción de HTTPS/WSS, rate limiting, y mejoras en la validación de entrada.
- Un análisis comparativo pre y post implementación, utilizando las mismas herramientas y metodologías del análisis inicial, que demostró la efectividad de las medidas implementadas.

Este enfoque mejoró significativamente la seguridad en el desarrollo de software moderno, abarcando desde el diseño hasta el mantenimiento continuo, y proporcionando una base sólida para la seguridad en aplicaciones web.

El estudio identificó tendencias clave en vulnerabilidades de sockets TCP, en los ataques de inyección de datos (35%), la denegación de servicio (48%), los problemas de gestión de sesiones (27%) y la falta de cifrado (40%). Se encontró que nuevas tecnologías web y vulnerabilidades particulares están relacionadas, lo que demuestra la necesidad de actualizaciones constantes en prácticas de seguridad. Las vulnerabilidades tradicionales se han vuelto más complejas, lo que requiere métodos de mitigación más avanzados.

La comparación de diferentes técnicas de análisis demostró que un enfoque combinado es efectivo. La revisión manual (97%), el análisis dinámico (92%), y el análisis estático demostraron fortalezas distintivas en diferentes etapas del desarrollo. El análisis estático se concentró en la detección temprana de problemas de codificación, mientras que el análisis dinámico se concentró en vulnerabilidades de tiempo de ejecución. Fuzzy

demonstró ser útil para encontrar vulnerabilidades no anticipadas, especialmente cuando se manejan entradas inesperadas.

Se desarrolló un marco de referencia integral, cuya eficacia se demostró con mejoras como HTTPS/WSS, que redujo los ataques de interceptación en un 98%, y rate limiting, que disminuyó los ataques de fuerza bruta en un 75%. El marco incorporó aspectos técnicos y de gestión, incluyendo revisiones de código enfocadas en seguridad y un proceso de mejora continua. Esto resultó en una mejora significativa en la calidad del código y en la capacidad de identificar y mitigar vulnerabilidades de seguridad en las aplicaciones de prueba.

RECOMENDACIONES

Se recomienda que futuras investigaciones en el campo de la seguridad de sockets TCP se enfoquen en el desarrollo de modelos predictivos basados en aprendizaje automático para la detección temprana de nuevas vulnerabilidades. Esto implicaría el análisis de grandes volúmenes de datos de tráfico de red y patrones de ataque, con el objetivo de anticipar y prevenir amenazas emergentes antes de que puedan ser explotadas.

Es crucial que los futuros estudios en este campo consideren la integración de técnicas de análisis de seguridad en las etapas tempranas del ciclo de desarrollo de software. Se sugiere investigar metodologías para incorporar el análisis estático y dinámico de manera automática en los procesos de integración y despliegue continuos (CI/CD), permitiendo una detección y mitigación más eficiente de vulnerabilidades en aplicaciones basadas en sockets.

Se aconseja explorar el desarrollo de frameworks especializados para la programación segura de sockets en diferentes lenguajes y entornos de desarrollo. Estos frameworks deberían incorporar las mejores prácticas de seguridad identificadas en esta investigación, facilitando a los desarrolladores la creación de aplicaciones robustas y seguras sin comprometer la funcionalidad o el rendimiento.

Se propone investigar la aplicación de técnicas de visualización avanzada para el análisis de vulnerabilidades en aplicaciones de sockets TCP. Esto podría incluir el desarrollo de herramientas que permitan a los analistas de seguridad y desarrolladores visualizar de manera intuitiva el flujo de datos, los puntos de entrada potenciales para ataques y las interacciones complejas entre diferentes componentes de la aplicación, mejorando así la comprensión y mitigación de riesgos de seguridad.

REFERENCIAS

- Abela, K. E. (2021). *Software security in web applications*. IGI Global.
- Anderson, R. (2008). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Indianapolis: Wiley.
- Axelsson, S. (2000). *Intrusion detection systems: A survey and taxonomy*. Chalmers University of Technology.
- Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C., & Shostack, A. (2002). Timing the application of security patches for optimal uptime. *LISA*, pp. 233-242.
- Boner, J. F. (2014). *Reactive Microsystems: The Evolution of Microservices at Scale*. O'Reilly Media.
- Burr, W. E., Dodson, D. F., Newton, E. M., Perlner, R. A., Polk, W. T., Gupta, S., & Nabbus, E. A. (2013). *Electronic authentication guideline*. National Institute of Standards and Technology.
- Chen, Y. W. (2022). Dynamic Analysis of TCP Socket Vulnerabilities in Modern Web Applications. *IEEE Transactions on Dependable and Secure Computing*, pp. 1782-1795.
- Clarke, J. (2009). *SQL Injection Attacks and Defense*. Burlington: Syngress.
- Comer, D. E. (2000). *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture*. Upper Saddle River: Prentice Hall.
- Doupé, A. C. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 111-131). Berlin, Heidelberg: Springer.
- Dowd, M., McDonald, J., & Schuh, J. (2006). *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Boston: Addison-Wesley Professional.

- Fette, I. &. (2011). *The WebSocket Protocol*. Retrieved from <https://tools.ietf.org/html/rfc6455>
- Fielding, R. &. (2014). *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. Retrieved from <https://tools.ietf.org/html/rfc7230>
- Frankel, S., & Krishnan, S. (2011). *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*. Internet Engineering Task Force.
- Garcia-Teodoro, P. D.-V.-F. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. In *Computers & Security* (pp. 18-28).
- Gupta, B. &. (2023). Comparative Analysis of Socket Security in RESTful and GraphQL APIs. *International Journal of Network Security*, pp. 168-179.
- Howard, M., & LeBlanc, D. (2003). *Writing Secure Code*. Redmond: Microsoft Press.
- IGM. (2017). *IGM*. Retrieved from Geoportal Ecuador – Infraestructura de Datos Espaciales: <https://www.geoportaligm.gob.ec/portal/>
- Koziol, J., Litchfield, D., Aitel, D., Anley, C., Eren, S., Mehta, N., & Hassell, R. (2004). *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Indianapolis: Wiley.
- Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach*. New York: Pearson.
- Lyu, M. R. (2019). A Comprehensive Study on Web Application Vulnerabilities. *IEEE*, pp. 154324-154342.
- Manico, J. A. (2017). *OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks*. Retrieved from <https://owasp.org/www-project-top-ten/2017/>
- McGraw, G. (2006). *Software Security: Building Security In*. Boston: Addison-Wesley Professional.
- Mirkovic, J., Dietrich, S., Dittrich, D., & Reiher, P. (2004). *Internet Denial of Service: Attack and Defense Mechanisms*. Upper Saddle River: Prentice Hall.

- Nakamura, Y. S. (2021). Fuzzing Socket APIs for Uncovering Vulnerabilities in Web Servers. *IEICE Transactions on Information and Systems*, pp. 2022-2032.
- Newman, S. (2015). *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc.
- Ornaghi, A., & Valleri, M. (2003). Man in the middle attacks demos. *Blackhat Conference Europe*.
- OWASP. (2021). *OWASP Top Ten*. Retrieved from <https://owasp.org/www-project-top-ten/>
- Parziale, L., Britt, D. T., Davis, C., Forrester, J., Liu, W., Matthews, C., & Rosselot, N. (2006). *TCP/IP Tutorial and Technical Overview*. Armonk: IBM Redbooks.
- Proxmox. (2024). *Proxmox*. Retrieved from Proxmox: <https://www.proxmox.com/en/>
- Rescorla, E. (2000). *SSL and TLS: Designing and Building Secure Systems*. Boston: Addison-Wesley.
- Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Engineering Task Force.
- Rescorla, E., & Modadugu, N. (2012). *Datagram Transport Layer Security Version 1.2*. Internet Engineering Task Force.
- Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, pp. 1278-1308.
- Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)*. National Institute of Standards and Technology.
- Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Chichester: Wiley.
- Sharma, V. V. (2020). A Systematic Review of Web Socket Security Vulnerabilities and Mitigation Techniques. *Journal of Information Security and Applications*, p. 102499.

- Srinivasan, S. N. (2020). JWT-Based Authentication and Authorization for Microservices. In *International Journal of Innovative Technology and Exploring Engineering* (pp. 1242-1246). 9.
- Srinivasan, S. N. (2020). JWT-Based Authentication and Authorization for Microservices. In *International Journal of Innovative Technology and Exploring Engineering* (pp. 1242-1246).
- Stallings, W. (2017). *Network Security Essentials: Applications and Standards*. New York: Pearson.
- Stevens, W. R., Fenner, B., & Rudoff, A. M. (2004). *UNIX Network Programming: The Sockets Networking API*. Boston: Addison-Wesley Professional.
- Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. Indianapolis: Wiley.
- Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks*. Boston: Prentice Hall.
- Tsyklevich, E., & Yee, B. (2003). Dynamic detection and prevention of race conditions in file accesses. *USENIX Security Symposium*.
- Yasar, H. &. (2016). Where to Integrate Security Practices on DevOps Platform. In *International Journal of Secure Software Engineering* (pp. 39-50).
- Ylonen, T., & Lonvick, C. (2006). The Secure Shell (SSH) Protocol Architecture. *Internet Engineering Task Force*.
- Zargar, S. T. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials*, pp. 2046-2069.

ANEXOS

Anexo 1: Configuración de servidor HTTPS con Socket.IO

```
import express from 'express';
import { createServer } from 'node:http';
import { Server } from 'socket.io';
import fs from 'node:fs';
import https from 'node:https';

const app = express();
const options = {
  key: fs.readFileSync('/etc/letsencrypt/live/sockets/privkey.pem'),
  cert: fs.readFileSync('/etc/letsencrypt/live/sockets/fullchain.pem')
};
const server = https.createServer(options, app);
const io = new Server(server, {
  connectionStateRecovery: {}
});

io.on('connection', (socket) => {
  console.log('Un cliente se ha conectado');
  // Resto del código
});

const port = process.env.PORT || 3000;
server.listen(port, () => {
  console.log(`Servidor seguro corriendo en https://localhost:${port}`);
});
```

Anexo 2: Implementación de rate limiting en Express.js

```
import express from 'express';
import rateLimit from 'express-rate-limit';

const app = express();

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100, // límite de 100 solicitudes por ventana por IP
  standardHeaders: true,
  legacyHeaders: false,
  message: 'Demasiadas solicitudes desde esta IP, por favor intente nuevamente después de 15 minutos'
});

app.use(limiter);

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => console.log('Servidor escuchando en el puerto 3000'));
```

Anexo 3: Función para sanitizar entrada de usuario en JavaScript

```
function sanitizeInput(input) {
  const map = {
    '&': '&amp;',
    '<': '&lt;',
    '>': '&gt;',
    '"': '&quot;',
    "'": '&#039;'
  };
  return input.replace(/[\&<>"]/g, function(m) { return map[m]; });
}
```

Anexo 4: Configuración básica de logging con Winston en Node.js

```
import winston from 'winston';

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  defaultMeta: { service: 'user-service' },
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple()
  }));
}
```

Anexo 5: Script de Python para pruebas de fuzzing en WebSocket

```
import websocket
import random
import string
import ssl

def random_string(length):
    return ''.join(random.choice(string.printable) for i in range(length))

def on_message(ws, message):
    print(f"Received: {message}")

def on_error(ws, error):
    print(f"Error: {error}")

def on_close(ws, close_status_code, close_msg):
    print("### closed ###")

def on_open(ws):
    print("Connected")
    for _ in range(1000): # Envía 1000 mensajes aleatorios
        payload = random_string(random.randint(1, 1000))
        ws.send(payload)
        print(f"Sent: {payload[:20]}...") # Muestra los primeros 20 caracteres

if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://192.168.245.115/",
                                on_open=on_open,
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)

    ws.run_forever(sslopt={"cert_reqs": ssl.CERT_NONE})
```


Anexo 6: Script de Python para análisis de tráfico WebSocket

```
from scapy.all import *

def packet_callback(packet):
    if packet.haslayer(TCP) and packet.haslayer(Raw):
        if packet[IP].src == "192.168.245.115" or packet[IP].dst == "192.168.245.115":
            print(f"Source: {packet[IP].src}:{packet[TCP].sport}")
            print(f"Destination: {packet[IP].dst}:{packet[TCP].dport}")
            print(f"Payload: {packet[Raw].load}")
            print("-----")

sniff(filter="tcp and host 192.168.245.115", prn=packet_callback, store=0)
```

Anexo 7: Script para Flooding

```
import asyncio
import socketio
import logging

# Configurar logging
logging.basicConfig(level=logging.DEBUG)

sio = socketio.AsyncClient(logger=True, engineio_logger=True)

@sio.event
async def connect():
    print('Conexión establecida')

@sio.event
async def connect_error(data):
    print(f'La conexión falló: {data}')

@sio.event
async def disconnect():
    print('Desconectado del servidor')

async def flood_chat():
    try:
        await sio.connect('http://localhost:3000', wait_timeout=10)
        for i in range(100):
            message = f"Test message {i}"
            await sio.emit('chat message', message)
            print(f"Enviado: {message}")
            await asyncio.sleep(0.1) # Envía un mensaje cada 100ms
    except Exception as e:
        print(f"Error durante el flooding del chat: {e}")
    finally:
        if sio.connected:
            await sio.disconnect()

async def main():
    print("Iniciando prueba de flooding para Chat")
    await flood_chat()
    print("Prueba de Chat completada")

if __name__ == "__main__":
    asyncio.run(main())
    return go(f, seed, [])
}
```