CIBERSEGURIDAD

Trabajo previo a la obtención de título de
Magister en Ciberseguridad

AUTOR/ES:

Enzo Leonardo Delgado Cadena

Merci Esmeralda León Morales

Sulema Verónica León Morales

Jorge Luis Mena Ludeña

TUTOR/ES:

Alejandro Cortés López

Iván Reyes Chacón

TEMA

Análisis y Evaluación de la Seguridad en una API RESTful para la gestión y reserva de máquinas en un gimnasio: Un Enfoque Basado en los Marcos NIST y OWASP API Security Top 10.

Quito - Ecuador

Enero – 2026

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, Delgado Cadena Enzo Leonardo, León Morales Merci Esmeralda, León Morales Sulema Verónica, Mena Ludeña Jorge Luis, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

Enzo Delgado
Digitally signed by Enzo Delgado
Date: 2026.01.11 22:11:52 -05'00'

Firma

Enzo Leonardo Delgado Cadena

Merci Leon
Digitally signed by Merci Leon
Date: 2026.01.11 22:12:10 -05'00'

Firma

Merci Esmeralda León Morales

Veronica Leon
Digitally signed by Veronica Leon
Date: 2026.01.11 22:12:29 -05'00'

Firma

Sulema Verónica León Morales

Jorge Mena
Digitally signed by Jorge Mena
Date: 2026.01.11 22:12:46 -05'00'

Firma

Jorge Luis Mena Ludeña

Autorización de Derechos de Propiedad Intelectual

Nosotros, Delgado Cadena Enzo Leonardo, León Morales Merci Esmeralda, León Morales Sulema Verónica, Mena Ludeña Jorge Luis, en calidad de autores del trabajo de investigación titulado ***Análisis y Evaluación de la Seguridad en una API RESTful para la gestión y reserva de máquinas en un gimnasio: Un Enfoque Basado en los Marcos NIST y OWASP API Security Top 10***, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, enero 2026

Enzo Delgado
Digitally signed by Enzo Delgado
Date: 2026.01.11 22:13:13 -05'00'

Firma

Enzo Leonardo Delgado Cadena

Merci Leon
Digitally signed by Merci Leon
Date: 2026.01.11 22:13:30 -05'00'

Firma

Merci Esmeralda León Morales

Veronica Leon
Digitally signed by Veronica Leon
Date: 2026.01.11 22:13:48 -05'00'

Firma

Sulema Verónica León Morales

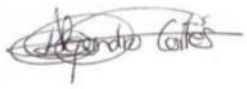
Jorge Mena
Digitally signed by Jorge Mena
Date: 2026.01.11 22:14:08 -05'00'

Firma

Jorge Luis Mena Ludeña

Aprobación de dirección y coordinación del programa

Nosotros, Alejandro Cortés López e Iván Reyes Chacón**,** declaramos que: Enzo Leonardo Delgado Cadena, Jorge Luis Mena Ludeña, Merci Esmeralda León Morales, Sulema Verónica León Morales son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés L.

Maestría en Ciberseguridad

Iván Reyes Ch.

Maestría en Ciberseguridad

DEDICATORIA

A Dios, por brindarnos la fortaleza, perseverancia y la claridad necesaria para culminar esta etapa de nuestra formación académica.

A nuestras familias, por su apoyo incondicional, comprensión y sobre todo paciencia durante el proceso de la maestría en las clases, siendo un pilar fundamental en cada desafío asumido.

A todas las personas que creen en el poder del conocimiento, saber, tecnología y la ciberseguridad como herramientas para proteger la información, fortalecer a las organizaciones y contribuir a una sociedad digital más segura.

Enzo, Merci, Verónica & Jorge

AGRADECIMIENTOS

De manera especial a la universidad que, por medio de la Facultad de Posgrado por la oportunidad de cursar la Maestría en Ciberseguridad, así como por los conocimientos impartidos que han fortalecido nuestra formación profesional.

Finalmente, a nuestros compañeros de maestría, de tesis y a todas las personas que, directa o indirectamente, contribuyeron con su apoyo, motivación y conocimientos durante este proceso académico.

Enzo, Merci, Verónica & Jorge

RESUMEN

Garantizar la seguridad de las aplicaciones web es un reto constante, especialmente con el manejo datos sensibles de los usuarios. Este proyecto de investigación tiene como objetivo evaluar de manera completa la seguridad de una API RESTful en una aplicación web para administrar usuarios y gestionar la reserva de máquinas de un gimnasio. Para llevar a cabo este trabajo se propone una metodología compuesta que se basa en el enfoque de gestión de riesgos del marco NIST con las directrices del OWASP API Security Top 10, posibilitando una evaluación de la API que contemple aspectos técnicos y metodológicos

La investigación comienza con la identificación detallada de los endpoints expuestos, análisis de tráfico, la revisión de la documentación y otras técnicas que nos permitan la recolección de información. Este paso inicial es clave para orientar las pruebas de seguridad posteriores. Luego, se realizan pruebas utilizando herramientas como Burp Suite y OWASP ZAP, complementadas con análisis manual, con el fin de revisar a fondo los mecanismos de autenticación, autorización, manejo de tokens, validación de parámetros y gestión de errores.

Una vez obtenidos los resultados, estos se organizan y clasifican según las categorías del OWASP API Security Top 10, se debe tomar en cuenta la posibilidad de que varias vulnerabilidades se relacionen entre sí y se podría realizar una explotación en conjunto. A partir de este análisis, se plantean recomendaciones técnicas orientadas a mejorar los controles de seguridad existentes y promover prácticas de desarrollo más seguras. Finalmente, se diseña un marco metodológico que resume de manera práctica todo el proceso utilizado, con el objetivo de que pueda aplicarse en futuras evaluaciones de seguridad de APIs en entornos similares.

PALABRAS CLAVES:

API RESTful, evaluación de seguridad, OWASP API Security Top 10, NIST Cybersecurity Framework, autenticación y autorización, control de acceso, gestión de reservas.

ABSTRACT

Guaranteeing the security of web applications remains a continuous challenge, especially when sensitive user data is involved. This research project aims to comprehensively evaluate the security of a RESTful API in a web application used to manage users and equipment reservations at a gym.

With this purpose, a composite methodology is proposed, based on the NIST framework's risk management approach and the OWASP API Security Top 10 guidelines. This approach enables an API evaluation that considers both technical and methodological aspects. The research begins with the detailed identification of exposed endpoints, traffic analysis, documentation review, and other techniques to gather information. This initial step is key to guiding subsequent security testing. Next, tests are performed using tools such as Burp Suite and OWASP ZAP, complemented by manual analysis, to thoroughly review the authentication, authorization, token handling, parameter validation, and error management mechanisms

Once the results are obtained, they are organized and classified according to the OWASP API Security Top 10 categories. It is important to consider the possibility that several vulnerabilities may be related and could be exploited together. Based on this analysis, technical recommendations are proposed to improve existing security controls and promote safer development practices. Finally, a methodological framework is designed that practically summarizes the entire process used, with the aim of enabling its application in future API security assessments in similar environments.

KEYWORDS:

RESTful API, security assessment, OWASP API Security Top 10, NIST Cybersecurity Framework, authentication and authorization, access control, reservation management.

CONTENIDO
0

LISTA DE TABLAS

LISTA DE FIGURAS

CAPITULO 1

1. INTRODUCCIÓN

1.1. Definición del proyecto

El presente proyecto tiene como objetivo la ejecución de un análisis de seguridad sobre una API RESTful, utilizada por una aplicación web construida y destinada a la gestión y reserva de máquinas en un gimnasio. Esta evaluación permitirá identificar vulnerabilidades críticas relacionadas con los mecanismos de autenticación, autorización, exposición de datos sensibles y control de acceso.

El análisis se basará en los lineamientos establecidos por el marco NIST (National Institute of Standards and Technology) para la gestión de riesgos de seguridad, complementado por el estándar OWASP API Security Top 10, que proporciona una guía práctica sobre las amenazas más frecuentes en APIs modernas.

La evaluación de seguridad se llevará a cabo mediante una combinación de herramientas automatizadas, como Burp Suite y OWASP ZAP, junto con técnicas manuales, con el fin de identificar vulnerabilidades que no pueden ser detectadas automáticamente. El proyecto incluirá también recomendaciones de mitigación para cada vulnerabilidad identificada, aportando un marco de buenas prácticas para el desarrollo seguro de APIs.

Este trabajo busca no solo evidenciar fallas en el entorno evaluado, sino también proponer un enfoque sistemático y reproducible que pueda ser aplicado a otros entornos reales, contribuyendo al fortalecimiento de la ciberseguridad en sistemas basados en APIs.

1.2. Justificación e importancia del trabajo de investigación

Hoy en día las API's constituyen parte de una columna vertebral del ecosistema digital, ya que prácticamente en su mayoría de lo que usamos en internet y en nuestros dispositivos móviles se basa en el uso de API's, estas exponen la lógica y datos de una organización al mundo exterior, según su configuración.

Es por esto, que aplicar técnicas de seguridad en el desarrollo e implementación de una API, tiene su importancia ya que esto permite mitigar la superficie de los riesgos relacionados con la fuga de información, interrupción del servicio o el compromiso total de los sistemas de la organización, al usar medidas de seguridad como la autenticación, autorización, cifrado, validación de datos de entrada, protección contra consumos excesivos y aplicación de configuraciones seguras, entre otras que hemos de enmarcar.

En el presente trabajo se abordarán los temas antes expuestos mediante la aplicación de pruebas de penetración que permitan identificar las vulnerabilidades que pudieran estar presentes en la API usando como referencia OWASP API Top 10, desde una óptica que permita definir y aplicar un estándar de evaluación dentro del proceso de certificación de seguridad previo al paso a producción de una API para verificar que las técnicas de seguridad definidas han sido aplicadas y no existan vulnerabilidades acorde al marco de pruebas, previo a su puesta en producción.

1.3. Alcance

El proyecto de investigación busca realizar una evaluación exhaustiva de la seguridad en la API de un gimnasio, la cual se utiliza para la gestión de usuarios y la reserva de máquinas. Para ello, se aplicará una metodología híbrida que combina el marco NIST para la gestión de riesgos con el estándar OWASP API Security Top 10 para identificar vulnerabilidades. El objetivo final es no solo encontrar debilidades, sino

también definir un estándar de pruebas de seguridad para APIs que pueda ser utilizado como guía antes de que estas salgan a producción.

1.4. Objetivos

1.4.1. Objetivo general

Evaluar de forma integral la seguridad de una API RESTful utilizada en una aplicación web para la gestión y reserva de máquinas de gimnasio, mediante la ejecución de pruebas de penetración automatizadas y manuales, con el fin de identificar vulnerabilidades técnicas relacionadas con autenticación, autorización, exposición de datos, validación de entradas y control de acceso, y proponer estrategias de mitigación basadas en los marcos OWASP API Security Top 10 y NIST SP 800-115, orientadas a mejorar la postura de seguridad de la solución tecnológica y establecer un marco replicable para futuras evaluaciones de API’s en entornos reales.

1.4.2. Objetivo especifico

- Identificar y documentar los endpoints expuestos de la API RESTful, a través de técnicas de recolección de información, análisis de tráfico y revisión de documentación técnica, con el propósito de establecer una base sólida para la ejecución de pruebas de seguridad focalizadas.
- Ejecutar pruebas de seguridad automatizadas y manuales sobre los diferentes componentes de la API, con énfasis en la evaluación de los mecanismos de autenticación, autorización, manejo de tokens, validación de parámetros y gestión de errores, utilizando herramientas como Burp Suite y OWASP ZAP, complementadas con técnicas manuales de análisis.
- Analizar e interpretar los hallazgos de seguridad obtenidos, clasificando las vulnerabilidades según su tipo y severidad de acuerdo con las categorías definidas

en el estándar OWASP API Security Top 10, considerando también posibles vectores de ataque combinados o cadenas de vulnerabilidades (vulnerability chaining).

- Evaluar el riesgo asociado a cada vulnerabilidad identificada, considerando tanto el impacto técnico como el contexto funcional de la aplicación, y apoyándose en criterios del marco NIST SP 800-115 para determinar niveles de criticidad y priorización de remediaciones.
- Proponer soluciones técnicas y recomendaciones de mejora, orientadas a mitigar las vulnerabilidades detectadas, fortaleciendo los controles de seguridad existentes y promoviendo prácticas de desarrollo seguro, con base en principios de arquitectura segura y controles compensatorios.
- Diseñar un marco metodológico práctico y replicable que consolide el proceso seguido durante la evaluación, incluyendo herramientas, criterios de análisis y pasos técnicos, con el objetivo de facilitar su aplicación en futuras auditorías de seguridad de API’s similares en otros entornos.

CAPITULO 2

2. REVISIÓN DE LITERATURA

2.1. Estado del Arte

Por varios años, la inclusión/adopción de las arquitecturas basadas en API's RESTful se ha incrementado notablemente debido a su capacidad para habilitar interoperabilidad, escalabilidad y modularidad en sistemas distribuidos especialmente. Estas interfaces se han convertido en un componente esencial más que crítico para aplicaciones móviles, o microservicios y plataformas de integración empresarial (Richardson & Amundsen, 2016). Sin embargo, este crecimiento exponencial también ha ampliado la superficie de ataque, posicionando a las API's como uno de los principales vectores de compromisos en la superficie de seguridad a nivel general.

Reportes recientes como *OWASP Top 10: API Security* (OWASP, 2023) y *2024 API Security Trends Report* (Salt Security, 2024) demuestran que más del 78% de los ataques modernos explotan vulnerabilidades en API's, sobre todo mal configuradas o autenticadas. En los escenarios de negocio que involucran reserva de recursos, gestión de usuarios o interacción en tiempo real como nuestra aplicación para gimnasio y/o servicios deportivos, suelen ser sensibles a ataques como Broken Object Level Authorization (BOLA), inseguridad en tokens JWT, fugas de información y fallas en mecanismos de rate limiting (Akhawe et al., 2020).

A nivel académico, los estudios de seguridad en las API's se han centrado en tres líneas:

1. **Detección y mitigación de vulnerabilidades específicas en APIs RESTful**, con enfoque en autenticación, autorización y validación (Gómez, Rojas & Crespo, 2021).

2. **Evaluación de APIs empleando modelos de riesgos**, principalmente NIST SP 800-53 y NIST Cybersecurity Framework (CSF), resaltando la necesidad de alineación con buenas prácticas de seguridad (Ross et al., 2020).
3. **Aplicación de los lineamientos OWASP API Security Top 10** en entornos de producción para fortalecer controles basados en riesgo (OWASP, 2023).

En este contexto, y expresado en el alcance dentro del capítulo 1, es necesario y relevante analizar la seguridad en una API RESTful, que está enfocada a la gestión y reserva de máquinas en un gimnasio, ya que combina autenticación de usuarios, acceso a datos personales y operaciones transaccionales, constituyendo un caso representativo para aplicar mecanismos de seguridad basados en los marcos NIST y OWASP.

2.2. Marco Teórico

2.2.1. Conceptos fundamentales de APIs RESTful

En el desarrollo actual y moderno de aplicaciones, las APIs RESTful se han convertido en una de las formas más utilizadas para permitir la comunicación entre sistemas y aplicaciones.

Una API RESTful es un mecanismo de comunicación entre aplicaciones que operan sobre el protocolo HTTP siguiendo los principios arquitectónicos del estilo REST (Representational State Transfer), definidos por Fielding (2000). REST establece seis restricciones clave: cliente-servidor, ausencia de estado (*stateless*), cacheabilidad, sistema por capas (*layered system*), interfaz uniforme y código bajo demanda (opcional).

Los elementos fundamentales son los recursos, los cuales se identifican mediante URLs y manipulan sus representaciones a través de métodos HTTP como

GET, POST, PUT y DELETE (Fielding & Taylor, 2000). Estas características han convertido a las APIs RESTful en una solución utilizada debido a su simplicidad, escalabilidad y adaptación natural al entorno web contemporáneo.

A continuación, se describe las características técnicas más relevantes:

- Statelessness: Cada petición debe contener toda la información necesaria para procesarse.
- Uniform Interface: Estandarización de operaciones mediante métodos HTTP.
- Representaciones estructuradas: Generalmente JSON o XML.
- Desacoplamiento entre cliente y servidor.
- Facilidad de escalado horizontal y compatibilidad con arquitecturas de microservicios.

La correcta implementación de estos principios es un prerrequisito para la seguridad, ya que malas prácticas como endpoints inconsistentes, exposición excesiva de recursos o manejo incorrecto del estado, facilitan ataques orientados a manipulación de objetos, fuga de información o abuso del API.

He aquí un amplio resumen de los códigos de estado HTTP:

Tabla 1 :

Codigo de estado HTTP

Código	Descripción
100-199	Respuestas informativas
200-299	Respuestas de éxito
300-399	Respuestas para redireccionamientos

400-499	Respuestas de error del lado del cliente
500-599	Respuestas de error del servidor[i]

Los códigos más utilizados en las API REST son en las áreas 200, 400 y 500. A continuación, algunos de los más comunes y útiles:

- **200**: Indica que la solicitud se ha realizado correctamente y no se ha creado ningún recurso. Se puede ver este código en respuesta a cosas como GET y PUT peticiones.
- **201:** Indica una solicitud correcta en la que se ha creado un recurso. Este suele ser el código de respuesta para POST solicitudes, aunque algunas POST no dan lugar a la creación de un recurso.
- **400:** Indica un error en la solicitud, normalmente con el cuerpo de la solicitud. Se puede utilizarse para indicar que la API no ha podido analizar el cuerpo de la solicitud.
- **401:** Indica que la solicitud no proporcionó la autenticación apropiada. Esto puede ocurrir cuando el cliente omite la autenticación o utiliza un token de autenticación obsoleto.
- **403:** Indica que un cliente autenticado no tiene acceso a la API o al recurso solicitado.
- **404:** Indica que el URI solicitado no corresponde a un punto final de la API.
- **500:** Indica un error interno del servidor. Se proporciona cuando el servidor encuentra un error desconocido.

Se pude encontrar una lista más completa de los códigos de estado HTTP, junto con las descripciones de cada uno, en la documentación de Mozilla sobre los códigos de estado de respuesta HTTP (Mozilla Developer Network, s.f.)

2.2.2. Amenazas Comunes en API's

Las API RESTful suelen presentar vulnerabilidades particulares debido a su exposición directa a internet y al procesamiento de información proveniente de usuarios, aplicaciones cliente y sistemas externos. Esta característica las convierte en un objetivo habitual para atacantes que buscan comprometer datos, modificar servicios

o explotar debilidades en la infraestructura. Entre los riesgos más relevantes se encuentran los descritos por organismos especializados en seguridad, como OWASP (OWASP Foundation, 2023).

2.2.3. OWASP API Security Top 10:

Open Web Application Security Project (OWASP) es una comunidad global que promueve prácticas abiertas para mejorar la seguridad de aplicaciones y API's. Su propuesta parte de la idea de que la protección de software no depende únicamente de herramientas técnicas, sino también del adecuado funcionamiento de los procesos organizacionales y de la participación de las personas involucradas (OWASP Foundation, 2023).

Uno de los elementos que distingue a OWASP es que es una comunidad de carácter independiente y sin fines de lucro, lo que le permite generar lineamientos y metodologías sin influencias comerciales. La organización se sostiene principalmente mediante la colaboración voluntaria de especialistas, líderes de capítulos locales y equipos de proyectos que contribuyen a la creación de recursos y guías de seguridad.

Dentro del panorama actual del desarrollo de software, las API's desempeñan un papel fundamental al facilitar la comunicación e integración entre sistemas y servicios distribuidos. No obstante, el ritmo acelerado con el que evolucionan estas tecnologías no siempre va acompañado de mecanismos de protección robustos, aumentando así su vulnerabilidad frente a ataques. En respuesta a esta problemática, OWASP desarrolló el **API Security Top 10**, un documento dirigido a profesionales del área como desarrolladores, arquitectos y administradores para apoyar la identificación y mitigación de los principales riesgos de seguridad en APIs (OWASP Foundation, 2023).

La versión publicada en 2023 recopila los diez riesgos más relevantes asociados con el diseño, desarrollo y operación de API's, ofreciendo explicaciones claras sobre su origen, su funcionamiento y las posibles consecuencias que pueden tener sobre la infraestructura tecnológica.

En la siguiente sección se describen los diez riesgos de seguridad definidos en el *OWASP API Security Top 10 (2023)*, destacando sus principales características, causas y el impacto que pueden tener sobre la seguridad de las API's.

2.2.3.1. API1:2023 Autorización de nivel de objeto roto

La autorización a nivel de objeto es un mecanismo de control de acceso que se implementa directamente en la lógica de la aplicación para garantizar que cada usuario solo pueda interactuar con los recursos que le corresponden. En cualquier endpoint que reciba un identificador de un recurso y ejecute acciones sobre él es imprescindible verificar que el usuario autenticado cuenta con permisos para realizar la operación solicitada.

Cuando estas validaciones no se aplican correctamente, pueden producirse accesos no autorizados que deriven en exposición, alteración o eliminación indebida de información.

Es importante señalar que simplemente comparar el ID del usuario obtenido de la sesión o de un token (como un JWT) con el identificador recibido en la petición no resuelve el problema conocido como Broken Object Level Authorization (BOLA), ya que este método solo cubre casos muy específicos.

En BOLA, el usuario puede acceder legítimamente al endpoint, pero el fallo ocurre porque puede manipular el identificador del recurso para operar sobre objetos ajenos. Por el contrario, si un atacante logra acceder a un endpoint que directamente no debería poder usar, el problema corresponde a una Broken Function Level Authorization (BFLA), y no a BOLA.

Ejemplo de escenarios de ataque

Un servicio de almacenamiento de documentos en línea permite a los usuarios ver, editar, almacenar y eliminar sus documentos. Cuando se elimina un documento de un usuario, se envía a la API una mutación de GraphQL con el ID del documento.

```
POST /graphql
{
 "operationName":"deleteReports",
 "variables":{
  "reportKeys":["<DOCUMENT_ID>"]
 },
 "query":"mutation deleteReports($siteId: ID!, $reportKeys: [String]!) {
  {
   deleteReports(reportKeys: $reportKeys)
  }
 }"
}
```

Dado que el documento con la ID dada se elimina sin más comprobaciones de permisos, un usuario podrá eliminar el documento de otro usuario.

2.2.3.2. API2:2023 Autenticación rota

Los endpoints y procesos de autenticación, incluidos los flujos de recuperación o restablecimiento de contraseña, deben considerarse componentes críticos y recibir el mismo nivel de protección que los mecanismos de inicio de sesión.

Una API se considera vulnerable cuando:

- Permite el uso de fuerza bruta para probar combinaciones de nombres de usuario y contraseñas válidas.
- No implementa bloqueos de cuenta, captchas u otras defensas ante intentos repetidos sobre la misma cuenta.
- Acepta contraseñas débiles o fáciles de adivinar.

- Envía información categorizada como sensible de autenticación (contraseñas o tokens) dentro de la URL, exponiéndola innecesariamente.
- Permite modificar datos críticos del usuario (correo, contraseña u otras acciones sensibles) sin requerir una confirmación adicional de identidad.
- No valida correctamente los tokens, permitiendo tokens sin firma o con algoritmos débiles, como JWT con algoritmo: "none".
- No verifica la fecha de expiración de los JWT.
- Almacena o maneja contraseñas en texto plano, con cifrado insuficiente o hashes débiles, o emplea claves de cifrado poco seguras.

Además, un microservicio es vulnerable si:

- Otros microservicios pueden acceder a él sin autenticación
- Utiliza tokens débiles o predecibles para imponer la autenticación

Ejemplos de escenarios de ataque

Para realizar la autenticación del usuario, el cliente debe emitir una solicitud API como la que se muestra a continuación con las credenciales del usuario:

```
POST /graphql
{
 "query":"mutation {
  login (username:\"<username>\",password:\"<password>\") {
   token
  }
  }"
}
```

Si las credenciales son válidas, se devuelve un token de autenticación, que debe proporcionarse en solicitudes posteriores para identificar al usuario. Los intentos de inicio de sesión están sujetos a una limitación de velocidad: solo se permiten tres solicitudes por minuto.

Para iniciar sesión por fuerza bruta en la cuenta de una víctima, los actores maliciosos aprovechan el procesamiento por lotes de consultas GraphQL para eludir la limitación de la tasa de solicitudes, lo que acelera el ataque:

```
POST /graphql
[
 {"query":"mutation{login(username:\"victim\",password:\"password\"){token}}"},
 {"query":"mutation{login(username:\"victim\",password:\"123456\"){token}}"},
 {"query":"mutation{login(username:\"victim\",password:\"qwerty\"){token}}"},
 ...
 {"query":"mutation{login(username:\"victim\",password:\"123\"){token}}"},
]
```

2.2.3.3. API3:2023 Autorización de nivel de propiedad de objeto roto

La categoría de Autorización de Nivel de Propiedad de Objeto Rota reúne fallas en las que un atacante accede a información sensible manipulando propiedades internas de los objetos en una API. Estos problemas suelen originarse en prácticas como la exposición excesiva de datos o la asignación masiva.

A diferencia de las vulnerabilidades que afectan al acceso a nivel de objeto completo, este riesgo se enfoca en controles más granulares, ya que un objeto puede contener atributos públicos y privados. Incluso cuando la API protege correctamente el acceso al objeto, puede seguir siendo vulnerable si no valida qué propiedades pueden consultarse o modificarse.

Ejemplo de escenario de ataque

Una app de citas permite a un usuario denunciar a otros usuarios por comportamiento inapropiado. Como parte de este proceso, el usuario hace clic en el botón "Denunciar" y se activa la siguiente llamada a la API:

```
POST /graphql
{
 "operationName":"reportUser",
 "variables":{
  "userId": 313,
  "reason":["offensive behavior"]
 },
 "query":"mutation reportUser($userId: ID!, $reason: String!) {
  reportUser(userId: $userId, reason: $reason) {
   status
   message
   reportedUser {
    id
    fullName
    recentLocation
   }
  }
 }"
}
```

El punto final de la API es vulnerable ya que permite que el usuario autenticado tenga acceso a propiedades de objetos de usuario confidenciales (informadas), como "fullName" y "recentLocation", a las que otros usuarios no deberían acceder.

2.2.3.4. API4:2023 Consumo de recursos sin restricciones

Las API's consumen recursos importantes del sistema como ancho de banda de red, CPU, memoria y almacenamiento. En ocasiones, los proveedores de servicios ponen a disposición los recursos necesarios mediante integraciones de API y se pagan por solicitud/demanda, como el envío de correos electrónicos, SMS, llamadas telefónicas, validación biométrica, etc.

Una API se considera expuesta cuando no define, o configura de manera incorrecta, los límites necesarios para controlar su consumo de recursos. Entre los parámetros más relevantes/comunes se encuentran:

- Límites de tiempo para la ejecución de operaciones.
- Cantidad máxima de memoria que puede utilizar una solicitud.
- Número permitido de descriptores de archivo y procesos simultáneos
- Número máximo de procesos
- Tamaño máximo de archivo de carga
- Número de operaciones a realizar en una única solicitud de cliente API (por ejemplo, procesamiento por lotes de GraphQL)
- Número de registros por página a devolver en una única solicitud-respuesta
- Límite de gasto de los proveedores de servicios externos

Ejemplos de escenarios de ataque

Una red social implementó un flujo de "olvidé mi contraseña" mediante verificación por SMS, lo que permite al usuario recibir un token único por SMS (OTP) para restablecer su contraseña.

Una vez que un usuario hace clic en "Olvidé mi contraseña", se envía una llamada API desde el navegador del usuario a la API de back-end:

```
POST /initiate_forgot_password
 {
 "step": 1,
 "user_number": "6501113434"
}
```

Luego, detrás de escena, se envía una llamada API desde el back-end a una API de terceros que se encarga de entregar los SMS:

```
POST /sms/send_reset_pass_code
 Host: willyo.net
 {
 "phone_number": "6501113434"
}
```

El proveedor externo, Willyo, cobra $0,05 por este tipo de llamada.

En un atacante se escribe un script que envía la primera llamada a la API miles de veces. El backend lo sigue y solicita a Willyo que envíe decenas de miles de mensajes de texto, esto provoca pérdidas de miles de dólares para la empresa en cuestión de minutos.

2.2.3.5. API5:2023 Autorización de nivel de función rota

La forma más efectiva de detectar fallas de autorización a nivel de función es analizar a fondo cómo opera el sistema de permisos, considerando la estructura de roles, grupos y privilegios dentro de la aplicación. Este análisis debe responder preguntas clave, como:

- Si un usuario con privilegios básicos puede acceder a funciones o endpoints administrativos.
- Si un usuario puede ejecutar acciones sensibles (crear, modificar o eliminar recursos) simplemente cambiando el método HTTP, por ejemplo, de GET a DELETE.
- Si un usuario de un grupo puede llegar a funciones destinadas exclusivamente a otro grupo adivinando rutas o parámetros, como /api/v1/users/export_all.

Además, no se debe asumir que un endpoint es administrativo solo por la ruta que utiliza. Aunque es habitual que las funciones de administración se agrupen bajo rutas como /api/admins, también es común encontrarlas mezcladas con rutas regulares como /api/users, lo que exige una verificación exhaustiva del control de acceso más allá del diseño del URL

Ejemplos de escenarios de ataque

Durante el registro en una aplicación que funciona mediante invitaciones, la app móvil realiza una solicitud a GET /api/invites/{invite_guid} para obtener información básica sobre la invitación, como el correo y el rol asignado.

Un atacante puede interceptar esa petición, duplicarla y modificar tanto el método HTTP como la ruta para enviarla a POST /api/invites/new, un endpoint reservado exclusivamente para administradores y accesible solo desde la consola interna. Debido a que este punto final no cuenta con controles de autorización a nivel de función, la API permite la operación pese a que el usuario no tiene privilegios administrativos.

El atacante aprovecha el problema y envía una nueva invitación con privilegios de administrador:

```
POST /api/invites/new
{
 "email": "attacker@somehost.com",
```

```
 "role":"admin"
}
```

Posteriormente, el atacante utiliza la invitación creada con fines maliciosos para crear una cuenta de administrador y obtener acceso completo al sistema.

2.2.3.6. API6:2023 Acceso sin restricciones a flujos comerciales sensibles

Al diseñar un endpoint de API, es fundamental entender el flujo de negocio que pone a disposición, ya que algunos procesos son especialmente sensibles y un acceso descontrolado puede generar impactos significativos.

Entre los flujos más delicados se encuentran aquellos que, si se abusan, pueden provocar:

- Compras masivas de productos en alta demanda, permitiendo prácticas como el scalping.
- Generación excesiva de contenido, facilitando envíos de spam en sistemas de comentarios o publicaciones.
- Bloqueo de reservas, donde un atacante ocupa todos los horarios disponibles para impedir el uso legítimo del servicio.

La criticidad de estos flujos depende del sector y del modelo de negocio: una acción que representa riesgo en una plataforma puede ser aceptable o incluso deseada en otra.

Un endpoint se considera vulnerable cuando expone un flujo comercial sensible sin aplicar controles de acceso adecuados, permitiendo que usuarios sin autorización ejecuten acciones que afectan la operación o disponibilidad del servicio.

Ejemplos de escenarios de ataque

Una empresa tecnológica anuncia el lanzamiento de una nueva consola de videojuegos el Día de Acción de Gracias. El producto tiene una gran demanda y el stock es limitado. Un atacante escribe código para comprar automáticamente el nuevo producto y completar la transacción.

El día del lanzamiento, el atacante ejecuta el código distribuido en diferentes direcciones IP y ubicaciones. Mientras la API no implementa una protección adecuada esto permite comprar la mayoría de las acciones antes que otros usuarios legítimos.

Posteriormente, el atacante vende el producto en otra plataforma por un precio mucho más alto.

2.2.3.7. API7:2023 Falsificación de solicitud del lado del servidor

Las vulnerabilidades de SSRF (Server-Side Request Forgery) aparecen cuando una API acepta una URL proporcionada por el usuario y realiza solicitudes externas sin validarla. Esto permite que un atacante haga que la aplicación contacte destinos no previstos, incluso aquellos que deberían estar protegidos por firewalls o redes internas.

El desarrollo moderno ha incrementado tanto la frecuencia como el impacto de este tipo de fallas. Por un lado, prácticas habituales como webhooks, descargas desde URL, vistas previas de enlaces o integraciones de SSO llevan a los desarrolladores a confiar en direcciones externas definidas por el usuario. Por otro, tecnologías como servicios en la nube, Kubernetes o Docker exponen interfaces de administración mediante rutas HTTP conocidas, lo que convierte estos puntos en objetivos atractivos para ataques SSRF.

Además, en entornos actuales es más complejo restringir el tráfico saliente de una aplicación, lo que amplía el riesgo de explotación. Si bien es difícil eliminar totalmente esta

amenaza, es fundamental escoger mecanismos de defensa que equilibren las necesidades del negocio con el nivel de exposición permitido.

Ejemplos de escenarios de ataque

Una red social permite a los usuarios subir fotos de perfil. El usuario puede elegir entre subir la imagen desde su ordenador o proporcionar la URL de la imagen. Al elegir esta última opción, se activará la siguiente llamada a la API:

```
POST /api/profile/upload_picture
{
 "picture_url": "http://example.com/profile_pic.jpg"
}
```

Un atacante puede enviar una URL maliciosa e iniciar un escaneo de puertos dentro de la red interna utilizando el punto final de la API.

```
{
 "picture_url": "localhost:8080"
}
```

Según el tiempo de respuesta, el atacante puede determinar si el puerto está abierto o no.

2.2.3.8. API8:2023 Configuración incorrecta de seguridad

La configuración incorrecta de seguridad es uno de los problemas más habituales en API's, ya que puede surgir en cualquier parte de la infraestructura cuando se dejan valores por defecto, se habilitan funciones innecesarias o no se gestionan adecuadamente los ajustes de servidores, redes o servicios en la nube.

Una API suele considerarse expuesta cuando presenta situaciones como:

- Ajustes mal configurados en la infraestructura o en servicios cloud, lo que deja permisos demasiado amplios o controles sin reforzar.
- Sistemas desactualizados o sin parches de seguridad aplicados.
- Funciones habilitadas sin necesidad, como verbos HTTP no usados o registros demasiado verbosos.
- Inconsistencias en el procesamiento de solicitudes entre servidores dentro de la cadena de manejo HTTP.
- Ausencia de cifrado en tránsito (TLS) o configuraciones inseguras en el canal de comunicación.
- Falta de cabeceras de seguridad, incluyendo políticas de caché, lo que expone información a clientes no confiables.
- CORS mal configurado, permitiendo accesos desde orígenes no autorizados.
- Mensajes de error demasiado detallados, que revelan información interna como trazas de pila o datos sensibles.

Ejemplos de escenarios de ataque

Un servidor back-end de API mantiene un registro de acceso generado por una popular utilidad de registro de código abierto de terceros, compatible con la expansión de marcadores de posición y búsquedas JNDI (Java Naming and Directory Interface), ambas habilitadas por defecto. Para cada solicitud, se escribe una nueva entrada en el archivo de registro con el siguiente patrón: <method> <api_version>/<path> - <status_code>.

Un actor malicioso emite la siguiente solicitud de API, que se escribe en el archivo de registro de acceso:

```
GET /healthX-Api-Version: ${jndi:ldap://attacker.com/xMalicious.class}
```

Debido a la configuración predeterminada insegura de la utilidad de registro y una política de salida de red permisiva, para escribir la entrada correspondiente en el registro de acceso, mientras se expande el valor en el X-Api-Versionencabezado de la solicitud, la utilidad de registro extraerá y ejecutará el Malicious.classobjeto desde el servidor controlado remotamente por el atacante.

2.2.3.9. API9:2023 Gestión inadecuada del inventario

Las API modernas forman ecosistemas amplios y conectados, lo que hace esencial que las organizaciones mantengan visibilidad completa sobre sus endpoints, versiones y los datos que manejan o comparten con terceros. Operar varias versiones de una misma API aumenta la carga de gestión y expande la superficie de ataque, por lo que se requiere un control riguroso.

Una API presenta un “punto ciego de documentación” cuando su propósito y estado no están claramente definidos: no se sabe en qué entorno opera, quién puede acceder a ella, qué versión está activa, o cuando la documentación es inexistente, está desactualizada o no existe un plan para retirar versiones antiguas. También ocurre cuando el inventario de hosts no está actualizado.

De igual forma, aparece un “punto ciego en el flujo de datos” cuando la API comparte información sensible con terceros sin una razón de negocio clara, sin registros que identifiquen ese flujo y sin saber exactamente qué datos se están exponiendo. Mantener un inventario preciso de estos movimientos de datos es clave para responder de manera efectiva ante incidentes de seguridad.

Ejemplos de escenarios de ataque

Una red social implementó un mecanismo de limitación de velocidad que impide a los

atacantes usar fuerza bruta para adivinar tokens de restablecimiento de contraseña. Mientras este mecanismo no se implementa como parte del código de la API, sino en un componente separado entre el cliente y una API oficial (api.socialnetwork.owasp.org). Un investigador encontró un host de API beta (beta.api.socialnetwork.owasp.org) que ejecuta la misma API, incluyendo el mecanismo de restablecimiento de contraseña, pero el mecanismo de limitación de velocidad no estaba implementado. El investigador pudo restablecer la contraseña de cualquier usuario usando fuerza bruta para adivinar el token de 6 dígitos.

2.2.3.10. API10:2023 Consumo inseguro de API

En muchos casos, los desarrolladores confían más en la información que reciben de API externas sobre todo si provienen de proveedores reconocidos, que en los datos ingresados por los usuarios. Esta confianza puede llevar a aplicar controles de seguridad, especialmente en la validación y el saneamiento de la información.

Una API puede volverse vulnerable cuando:

- Se comunica con servicios externos sin cifrado, exponiendo la información en tránsito.
- Procesa datos de terceros sin validarlos ni limpiarlos, permitiendo que información maliciosa avance dentro del sistema.
- Acepta redirecciones sin verificación, lo que puede guiar la aplicación hacia destinos inesperados.
- No controla los recursos utilizados para manejar respuestas de otros servicios, lo que podría permitir abusos o consumo excesivo.
- No define tiempos de espera, dejando la aplicación bloqueada o consumiendo recursos mientras espera respuestas externas.

Ejemplos de escenarios de ataque

Una API se integra con un proveedor de servicios externo para almacenar de forma segura la información médica confidencial del usuario. Los datos se envían a través de una conexión segura mediante una solicitud HTTP como la siguiente:

```
POST /user/store_phr_record
{
 "genome": "ACTAGTAG__TTGADDAAIICCTT…"
}
```

Los malos actores encontraron una forma de comprometer la API de terceros y esta comienza a responder con 308 Permanent Redirectsolicitudes como la anterior.

HTTP/1.1 308 Permanent Redirect

Location: https://attacker.com/

Dado que la API sigue ciegamente las redirecciones de terceros, repetirá exactamente la misma solicitud, incluidos los datos confidenciales del usuario, pero esta vez al servidor del atacante.

Controlar redirecciones: Mantener un listado de destinos confiables y no seguir redirecciones de manera automática evita que la API sea explotada para enviar solicitudes a sitios maliciosos.

2.2.4. Otros Riesgos de Seguridad en APIs:

Además de las vulnerabilidades ampliamente reconocidas en estándares como OWASP API Security Top 10, existen otros riesgos emergentes que afectan de forma significativa la seguridad de las API's actuales. Estos riesgos suelen originarse por prácticas deficientes de gobernanza, errores de diseño, falta de visibilidad del inventario de servicios y el uso creciente de integraciones con terceros. A continuación, se citan algunos de los más

relevantes, cuyo impacto puede comprometer tanto la integridad como la disponibilidad de los sistemas expuestos**.**

2.2.4.1. Minería de API REST para detectar posibles vulnerabilidades de asignación masiva

Los desarrolladores en algunos casos suelen depender de frameworks para construir API's REST. Estos frameworks, como Spring para Java, Flask para Python, Express.js para JavaScript y Laravel para PHP, ofrecen un conjunto de componentes reutilizables y funciones que facilitan el desarrollo de las API's REST. Una de las funcionalidades que suelen proporcionar estos frameworks se denomina auto-binding (auto-vinculación), un mecanismo que adopta convenciones de nombres para mapear automáticamente los datos de entrada en las solicitudes HTTP (es decir, los parámetros) a los objetos de datos del backend (por ejemplo, columnas de una base de datos) cuando comparten el mismo nombre.

Esta función suele estar habilitada por defecto para todos los atributos. Una vulnerabilidad de asignación masiva, también conocida como object injection o auto-binding vulnerability, ocurre cuando los desarrolladores no deshabilitan esta función para los atributos que deben ser de solo lectura. Como resultado, un atacante puede añadir un atributo adicional a una solicitud HTTP (uno que no se tenía intención de permitir modificar), y la función de auto-binding enlazaría automáticamente ese atributo con su representación interna correspondiente, por ejemplo, una columna de base de datos. En principio, dicho atributo ni forma parte de la especificación de la API ni de su documentación, y no debería haber sido procesado. Sin embargo, un atacante que explota esta funcionalidad podrá manipular y alterar datos en la base de datos, representando un riesgo de seguridad significativo.

2.2.4.2. API de Shadow & Zombie: puntos finales ocultos

Las API zombi son puntos finales que antes se usaban, pero que ahora están obsoletos, en desuso u olvidados. Pueden permanecer en su entorno mucho tiempo después de ser necesarias, lo que expone a posibles vulnerabilidades.

Las API ocultas son aquellas creadas sin el conocimiento, ni la aprobación de los equipos de seguridad. Suelen surgir cuando los equipos de desarrollo crean nuevos endpoints sin documentarlos ni integrarlos adecuadamente en los procesos formales de gestión de API.

Las herramientas de seguridad suelen pasar por alto ambos tipos de API, pero son objetivos fáciles para los atacantes. Sin supervisión ni gestión, estos endpoints pueden proporcionar puntos de acceso fáciles para filtraciones de datos, accesos no autorizados y otras actividades maliciosas.

Para prevenir la asignación masiva, los desarrolladores deben poner en lista negra los atributos de solo lectura para evitar que sean auto-vinculados a la representación interna de datos de las API's.

2.2.4.3. Riesgos generados por la IA generativa: expansión de la superficie de ataque y fuga de datos

La IA generativa (GenAI) incrementa de forma sustancial la superficie de ataque de las API al habilitar vectores de amenaza automatizados y altamente escalables. Los modelos generativos pueden producir patrones de tráfico malicioso, fuzzing inteligente y secuencias complejas de solicitudes que facilitan la identificación y explotación de vulnerabilidades, lo que eleva la probabilidad de ataques de denegación de servicio (DoS) y de compromisos a nivel lógico.

Así mismo, los extensos conjuntos de datos utilizados para entrenar modelos GenAI introducen riesgos inherentes de fuga de información sensible, ya sea mediante respuestas generadas, extracción inversa de datos o abuso de endpoints expuestos. La mitigación de estos escenarios requiere la implementación de mecanismos de autenticación y autorización reforzados, monitoreo continúo basado en telemetría avanzada y un enfoque de defensa en profundidad que combine controles tradicionales con técnicas de detección y respuesta asistidas por IA

2.2.4.4. Inyección inmediata: ataques dirigidos a puntos finales controlados por LLM

La inyección rápida es un ataque novedoso que apunta a modelos de lenguaje grandes (LLM) a los que se accede a través de API. Los atacantes insertan instrucciones dañinas en los mensajes, engañando al modelo para que se comporte de manera impredecible, como filtrar datos confidenciales o ejecutar acciones dañinas a través de complementos.

Estos ataques se dirigen a la capa de aplicación que utiliza LLM, no al modelo de IA central en sí, y pueden usarse para eludir las protecciones previstas. Una sólida desinfección de entrada, un filtrado rápido y un diseño de complementos LLM que tenga en cuenta la seguridad pueden reducir los riesgos.

2.2.4.5. Vulnerabilidades de la API GraphQL: Detección de inyección de consultas maliciosas

Las API GraphQL permiten consultas flexibles, pero esta flexibilidad puede generar consultas complejas y maliciosas que provoquen sobrecarga o fugas de datos. Los atacantes

crean consultas anidadas o costosas para inducir una denegación de servicio o inyectar cargas maliciosas, como inyecciones de SQL o de comandos.

La seguridad de API tradicional es insuficiente; la detección de anomalías impulsada por IA y el aprendizaje automático pueden ayudar a detectar y bloquear consultas maliciosas en tiempo real. Las protecciones básicas incluyen la limitación de la profundidad de las consultas, la limitación de la velocidad y el monitoreo de patrones de consultas inusuales.

2.2.5. Marcos de Gestión de Riesgos de Seguridad

Los marcos de ciberseguridad como NIST, han sido adoptados por diferente tipo de organizaciones ya sean estas pymes, grandes, o emprendedores para gestionar los riegos en sus apps, inclusos en sus sistemas de información, y más aún cuando el manejo está en API's.

2.2.5.1. NIST Cybersecurity Framework (CSF)

Proporciona un enfoque estructurado basado en cinco funciones: Identify, Protect, Detect, Respond y Recover, permitiendo evaluar riesgos y controles aplicables a componentes como APIs (NIST, 2018). En el contexto de una API:

- **Identify:** inventario de endpoints, recursos, tokens y flujos de negocio.
- **Protect:** aplicación de controles de acceso, cifrado, rate limiting y medidas anti-abuso.
- **Detect:** monitoreo de consumo anómalo, ataques de fuerza bruta o patrones de explotación.
- **Respond:** activación de alertas, contención de incidentes y revocación de tokens.
- **Recover:** restauración de la operación segura, generación de parches y lecciones aprendidas.

NIST SP 800-53 Rev. 5

Define un catálogo exhaustivo de controles de seguridad y privacidad, organizados por familias (AC, AU, SC, SI, etc.) que pueden aplicarse directamente a APIs, como:

AC-3 — Control de acceso.

SC-12 — Protección criptográfica.

SI-10 — Validación de entradas.

AU-6 — Auditoría y monitoreo.

(Ross et al., 2020).

NIST SP 800-30

Establece la metodología oficial para realizar análisis de riesgos, permitiendo identificar amenazas, vulnerabilidades, impactos y probabilidad en sistemas basados en APIs (NIST, 2012).

La combinación de NIST y OWASP permite un enfoque híbrido:

NIST → Gestión del riesgo, controles y madurez.

OWASP → Vulnerabilidades técnicas específicas de APIs.

Marco NIST:

El Instituto Nacional de Estándares y Tecnología (NIST) establece una serie de lineamientos y marcos de referencia ampliamente utilizados para la gestión de riesgos en sistemas de información. Entre ellos, destacan el NIST Risk Management Framework (RMF) (NIST SP 800-37), el NIST Cybersecurity Framework (CSF) y los controles de seguridad de

NIST SP 800-53 Rev. 5, que proporcionan un enfoque sistemático para identificar, evaluar y mitigar riesgos en infraestructuras tecnológicas, incluyendo APIs RESTful.

Metodologías de Pentesting de APIs:

El pentesting de APIs requiere enfoques específicos debido a su naturaleza orientada a recursos, su dependencia en métodos HTTP y su exposición directa a usuarios y automatizaciones. Las metodologías más relevantes se basan en estándares reconocidos como OWASP, NIST y PTES.

OWASP API Security Testing Guidance

OWASP provee un marco especializado para pruebas en APIs, que incluye:

- Identificación de endpoints explícitos y ocultos
- Enumeración de métodos y parámetros
- Pruebas de autenticación (tokens JWT, OAuth2)
- Pruebas de autorización (BOLA/BFLA)
- Manipulación de objetos y validación de parámetros
- Pruebas de rate limiting y abuso de recursos
- Validación de configuraciones del servidor, CORS y TLS

Este enfoque se alinea con el OWASP API Security Top 10 (2023) y permite evaluar los riesgos más críticos.

NIST Technical Guide to Information Security Testing (SP 800-115)

NIST SP 800-115 proporciona directrices para ejecutar pruebas de seguridad técnicas, organizadas en:

1. Planificación: reglas de compromiso, alcance de la API, datos sensibles.

2. Descubrimiento: recolección de información y enumeración mediante fuzzing, inspección del código y pruebas dinámicas.
3. Ataque: explotación de endpoints inseguros, pruebas de autenticación, autorización y validación.
4. Reporte: registro de hallazgos, evidencia, impacto y recomendaciones.

Aunque SP 800-115 no está centrado únicamente en APIs, sus principios se adaptan con facilidad al entorno API-driven.

PTES (Penetration Testing Execution Standard)

PTES define fases estructuradas: inteligencia, modelado de amenazas, análisis, explotación, post-explotación y reporte.

En APIs RESTful, se aplica en:

- Modelado de amenazas basado en flujos del API
- Explotación mediante manipulación de recursos
- Evaluación de impacto en la lógica de negocio

El uso conjunto de OWASP + NIST SP 800-115 + PTES ofrece un enfoque híbrido que cubre tanto amenazas técnicas como riesgos de negocio, como se lo plantea en el alcance de este documento.

Consideraciones Regulatorias en Ecuador:

Durante la ejecución del análisis se consideran las normativas nacionales e internacionales aplicables a la protección de datos personales:

- Ley Orgánica de Protección de Datos Personales (LOPD, Ecuador, 2021): exige consentimiento expreso y medidas de seguridad para la recolección y tratamiento de información de los usuarios del gimnasio.
- Código Orgánico Integral Penal (COIP, Ecuador): establece sanciones por acceso no autorizado a sistemas o datos sensibles.
- Reglamento General de Protección de Datos (GDPR, UE) como referencia internacional para principios de minimización de datos, transparencia y responsabilidad.

El pentesting respeta estas normativas mediante la anonimización de datos sensibles, la ejecución de pruebas en entornos controlados y la ausencia de exfiltración no autorizada de información.

La evaluación de seguridad de la API se realiza considerando las normativas ecuatorianas e internacionales relacionadas con la protección de datos personales y el acceso no autorizado a sistemas de información.

Ley Orgánica de Protección de Datos Personales (LOPDP, Ecuador, 2021)

La LOPDP (2021) establece los principios esenciales para el tratamiento adecuado de datos personales en Ecuador, entre ellos el consentimiento informado, la minimización de datos, la seguridad, la confidencialidad y la responsabilidad proactiva.

Para una API que gestiona reservas en un gimnasio, la ley exige:

- Contar con consentimiento expreso del usuario para el tratamiento de datos personales.
- Implementar medidas técnicas y organizativas adecuadas (Art. 42–45), tales como cifrado, autenticación fuerte y controles de acceso.
- Evitar la recopilación excesiva de información (Art. 7 – Minimización).

Código Orgánico Integral Penal (COIP, Ecuador)

El COIP tipifica delitos relacionados con acceso no autorizado, manipulación, interceptación o destrucción de información (Art. 230–235).

Durante un pentest se deben evitar acciones que:

- Accedan a información fuera del alcance autorizado.
- Alteren bases de datos o recursos del sistema.
- Intercepten comunicaciones sin autorización.

El pentesting permitido debe operar bajo reglas de compromiso (RoE) firmadas y ambientes controlados.

Reglamento General de Protección de Datos (GDPR, UE)

Aunque no es obligatorio en Ecuador, el GDPR sirve como marco internacional de referencia para buenas prácticas de protección de datos, especialmente en:

- Minimización (Art. 5)
- Transparencia (Art. 12)
- Seguridad del tratamiento (Art. 32)

- Evaluación de impacto (Art. 35)

La consideración del GDPR permite alinear la evaluación del API con estándares globales de privacidad.

Cumplimiento durante el Pentesting

El pentesting se ejecuta aplicando los siguientes principios regulatorios:

- Anonimización o uso de datos ficticios para las pruebas.
- Ejecución en entornos controlados sin afectar sistemas productivos.
- No exfiltrar datos reales, incluso si una vulnerabilidad lo permite.
- Registro documental de cada acción para demostrar transparencia y diligencia.

Estas prácticas aseguran que la evaluación de seguridad cumpla con la LOPDP, COIP y principios del GDPR.

CAPITULO 3

3. DESARROLLO E IMPLEMENTACIÓN DEL ENTORNO

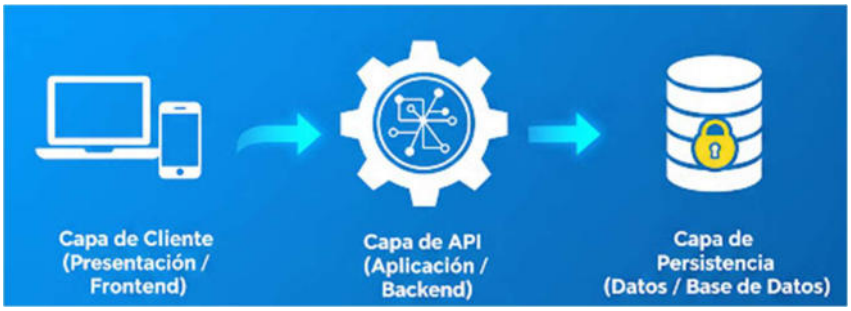
3.1. Descripción de la arquitectura del sistema informático

3.1.1 Vista general de la arquitectura

El sistema de información de GymMax está diseñado bajo una arquitectura cliente-servidor moderna, es decir utilizando específicamente un diseño de tres capas. Esta estructura separa de manera rigurosa la lógica del negocio de la interfaz de usuario. A continuación, en la figura 1 se presenta la arquitectura del sistema informatico

Figura 1:

Arquitectura del sistema informático



3.1.2 Componentes y Capas

El sistema informático se estructura en una arquitectura de tres capas bien definidas, la capa de presentación se ejecuta en el navegador web del cliente utilizando tecnologías como: HTML5, CSS3, y JavaScript vanilla (incluyendo LocalStorage API), y se comunica con la siguiente capa a través del protocolo HTTPS.

La capa de aplicación es la esencia de la funcionalidad del sistema. Implementada como una API RESTful, es la encargada de ejecutar la lógica del negocio y gestionar la autenticación de usuarios y asegurar la correcta validación de los datos.

La API se interconecta a la capa de persistencia para el almacenamiento seguro y persistente de los datos, además de establecer comunicación mediante el protocolo TCP. A continuación, se presenta una tabla detallando las capas y sus componentes.

Tabla 2:

Capas y Componentes de la Arquitectura

Capa	Componente	Tecnología Clave	Ubicación y Comunicación
1.Presentación	Clientes / Frontend	HTML5, CSS3, JavaScript vanilla (ES6+), LocalStorage API	Se ejecuta en el Navegador Web. Envía peticiones a la API a través de Fetch API y HTTPS.
2. Aplicación	API RESTful (Backend)	Node.js + Express	Alojada en VM 2 (192.168.56.20), expuesta por el Puerto 3000 (HTTPS). Es responsable de la autenticación, validación de datos y lógica de negocio.

3. Persistencia	Base de Datos	PostgreSQL	Alojada en VM 1 (192.168.56.10), accesible por el Puerto 5432 a través de TCP.

3.1.3 Flujo de Datos

El flujo de datos describe el recorrido que realiza una petición hasta su respuesta dentro del sistema.

Petición del cliente: El usuario final (cliente o empleado) interactúa con las páginas HTML a través de los formularios. JavaScipt realiza una validación inicial y construye una solicitud JSON que se envía vía HTTPS a la API.

Usuario → HTML Form → JavaScript (Validación) → Fetch API → HTTPS → Backend

Procesamiento en la API: La API (EXPRESS server) recibe la petición, donde realiza la validación de entrada, aplica la lógica de negocio y verifica la autenticación. Si se requiere de información, se genera una Query y se comunica con la base de datos a través de TCP.

Consulta a la base de datos: PostgreSQL procesa la consulta sql y devuelve el resultado a la API.

Respuesta al cliente: La API genera una respuesta en formato JSON y la retorna de vuelta al cliente mediante HTTPS. JavaScript en la parte del frontend parsee la respuesta JSON y actualiza el HTML para presentar el resultado al usuario.

3.2. Infraestructura de Hosting

La infraestructura del sistema de información se implementa mediante un entorno de máquinas virtuales (VM) segmentadas en una red local privada, asegurando una separación de los componentes clave: API y la base de datos.

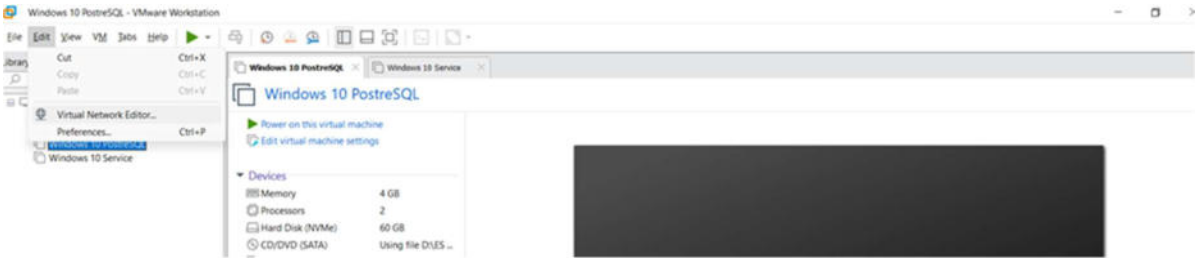
Configuración de la red privada en VMware

Para la configuración de red se utilizó el modo Host Only Network, lo que permite la comunicación de las máquinas virtuales y el host, además de aislarlas de la red física exterior. El procedimiento realizado para dicha configuración es el siguiente.

En el virtualizador VMware Workstation, nos dirigimos al menú Edit > Virtual Network Editor (Editor de Redes Virtuales). En la figura 2 nos muestra el Editor de Redes Virtuales

Figura 2:

Editor de Redes Virtuales

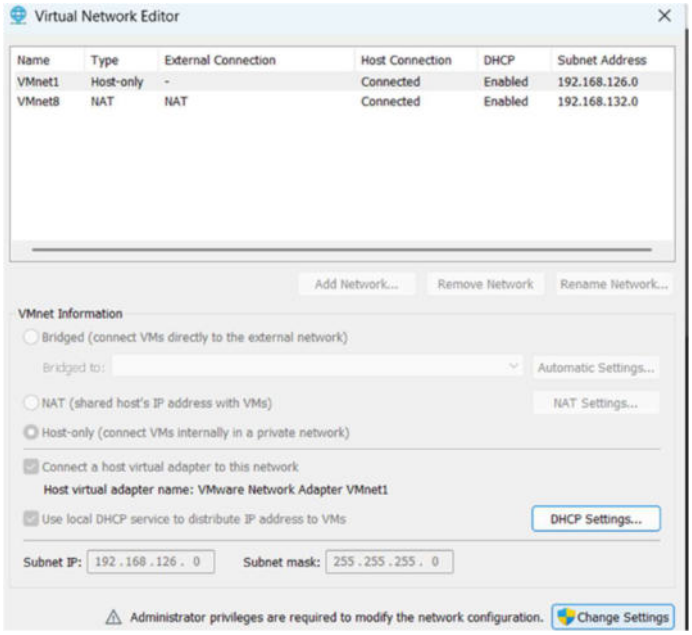


De ser necesario, hacemos clic en Change Settings para obtener permisos de administrador. Esto se muestra en la figura 3

.

Figura 3:

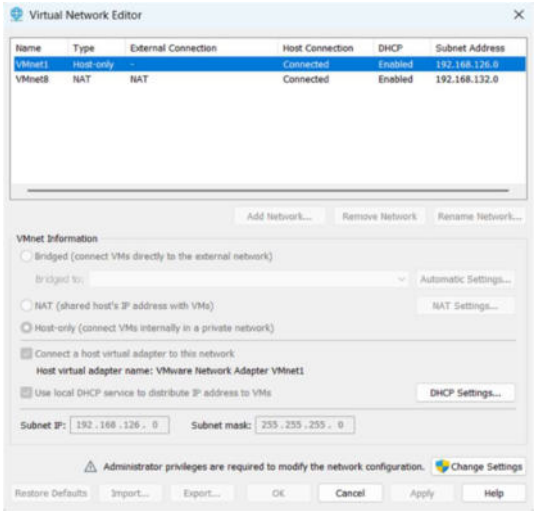
Obtención de Permisos de Administrador



Buscamos la red VMnet1, que está configurada en modo Host Only. Si no existe se puede añadir una nueva dando clic en add y seleccionamos el modo deseado. La figura 4 muestra la selección o creación de la red

Figura 4:

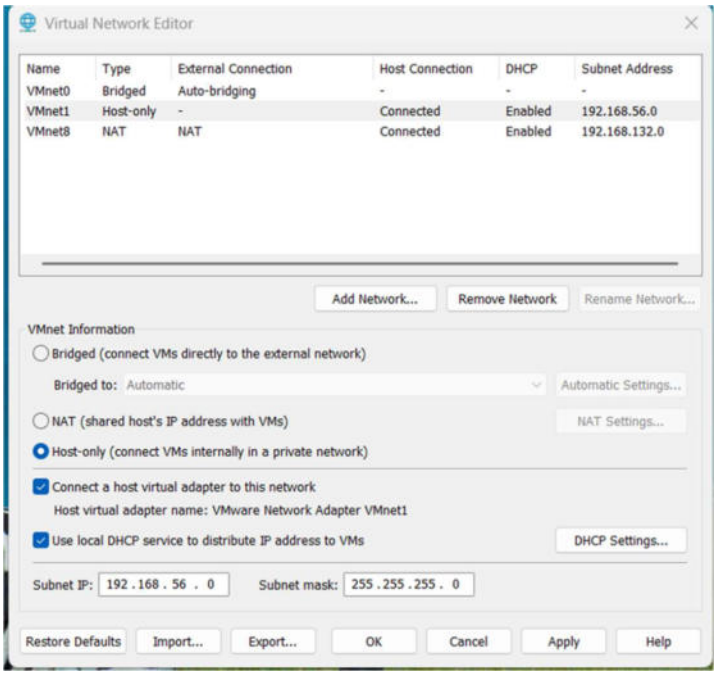
Selección o creación de la red



Definiremos y configuraremos la subred, en este caso será la 192.168.56.0

Figura 5:

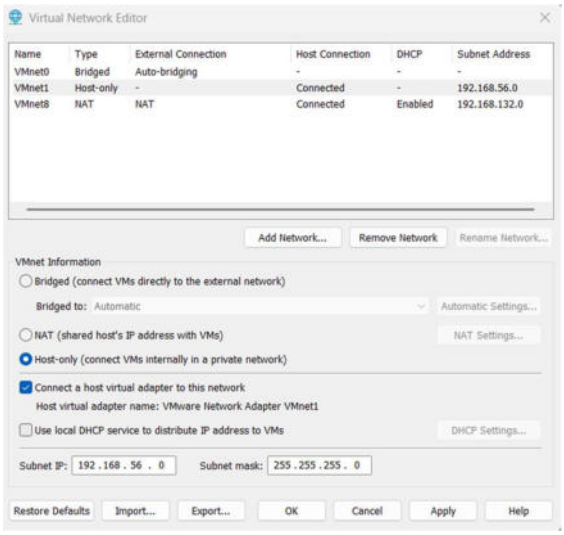
Definición y configuración de la subred



Deshabilitaremos el protocolo DHCP que nos da el vitualizador VMware ya que asignaremos ip's estáticas para nuestras máquinas virtuales.

Figura 6:

Desactivación del DHCP



Aplicamos el modo host only a nuestras máquinas virtulales para lo cual ambas máquinas deben estar apagadas, después nos dirigimos a settings(configuraciones) y selecionamos el modo deseado para el adaptador de red. A continuación,véase en la figura 7

Figura 7:

Modo host only para el adaptador de red máquina PostreSQL

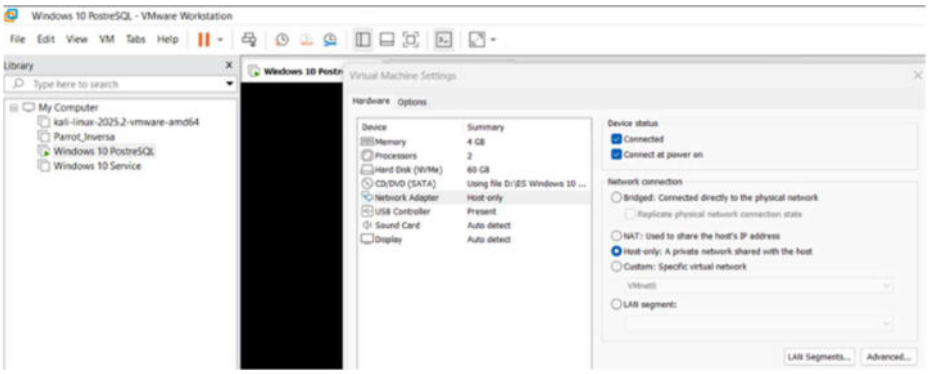
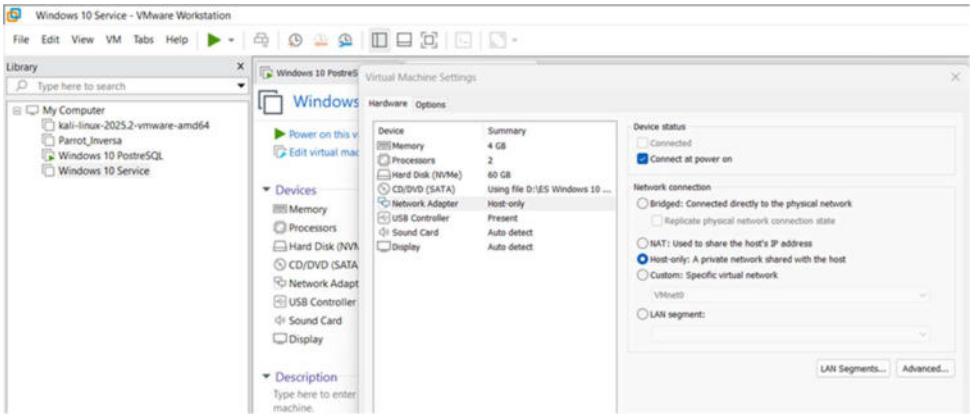


Figura 8:

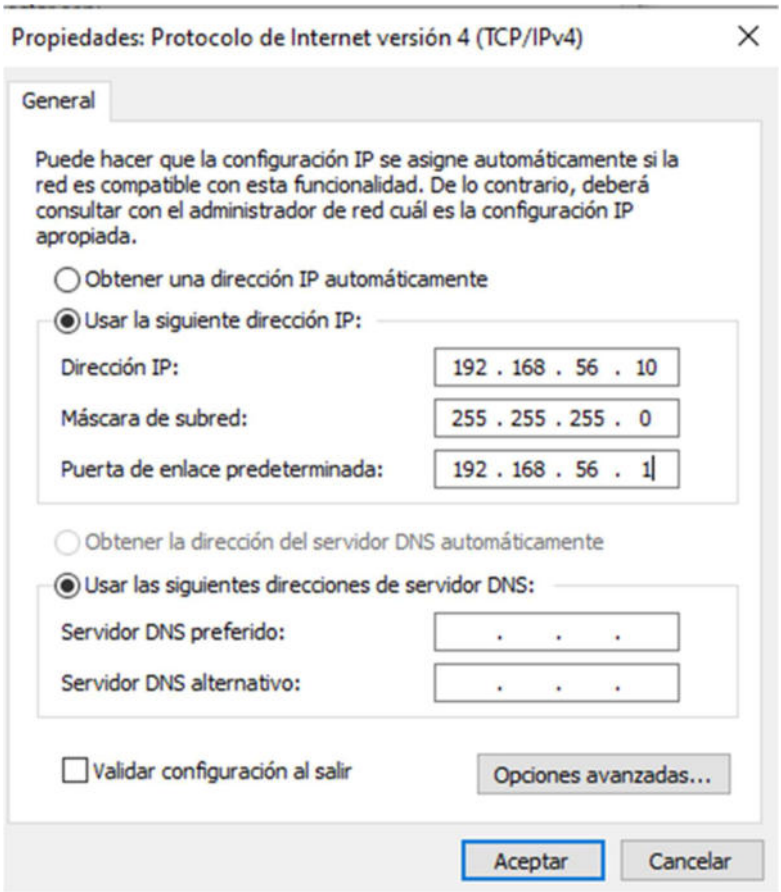
Modo host only para el adaptador de red máquina Service



Colocamos las direcciones ip estáticas a nuestras máquinas virtuales, para la máquina PostreSQL la dirección asiganda será 192.168.56.10. Como Muestra en la figura 9

Figura 9:

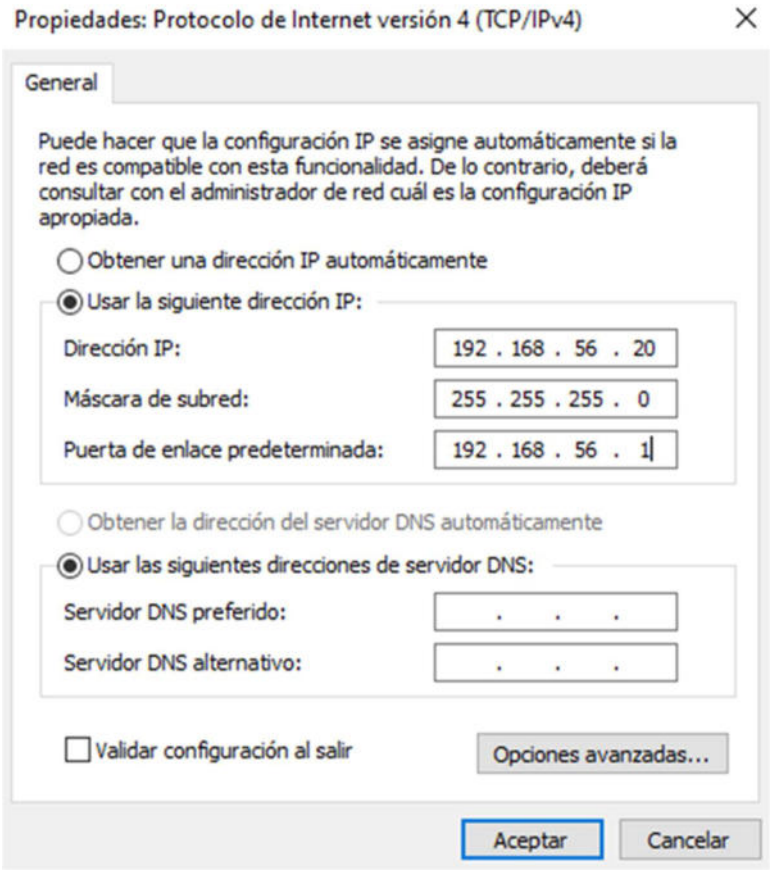
Asignación de dirección ip 192.168.56.10



Para la máquina Service que es en donde se estará ejecutando nuestro api, le asignaremos la dirección ip estática 192.168.56.20. Ver figura 10.

Figura 10:

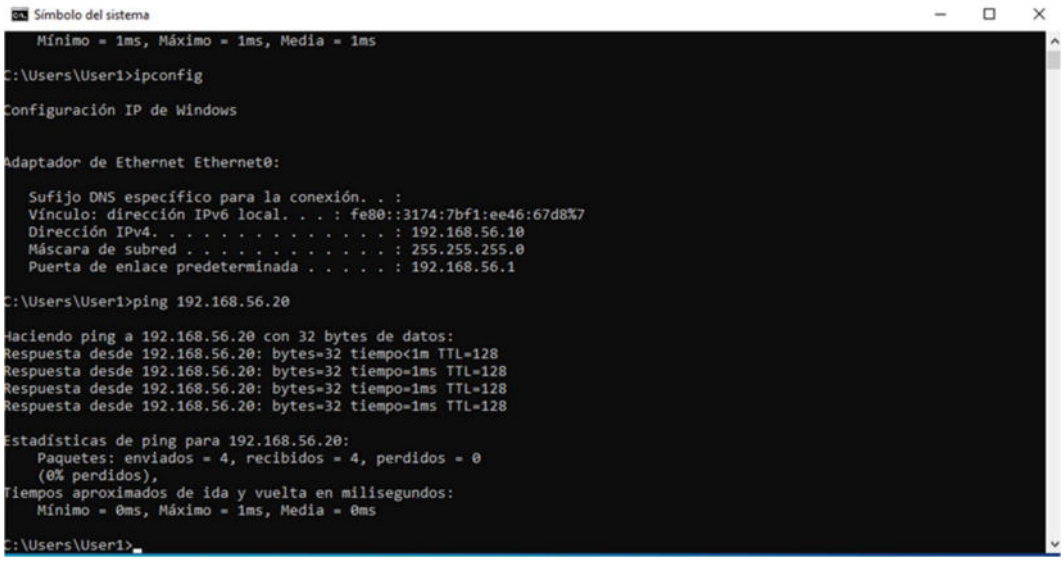
Asignación de dirección ip 192.168.56.20



Probamos la conectividad de las máquinas haciendo ping desde Windows PostreSQL a Windows Service. Prueba de conectividad en la figura 11

Figura 11:

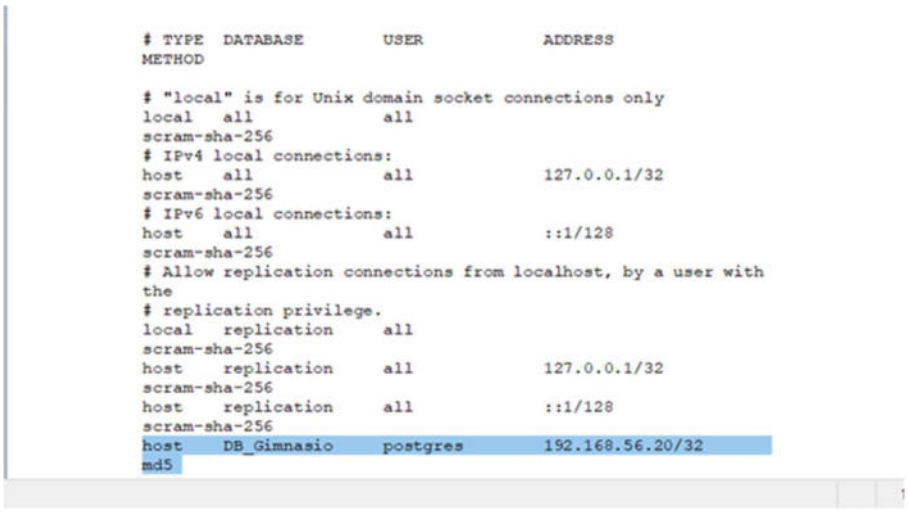
Prueba de conectividad



Para levantar el servicio, lo primero que debemos hacer es crear una regla de entrada para el puerto 5432 en la máquina virtual que contiene la base de datos para permitir la conexión desde la subred 192.168.56.0/24.Dicha regla se agregará en el fichero pg_hba.conf localizado en: C:\Program Files\PostgreSQL\16\data\pg_hba.conf con la estructura que se detalla en la figura12.

Figura 12:

Regla de acceso



Distribución de red y puertos

La arquitectura del host define una clara segmentación y control de acceso mediante puertos. En cuanto a la Seguridad Perimetral y de Transporte, se implementa el uso obligatorio de HTTPS (SSL/TLS) en los puertos 3000 y 443 (el puerto de desarrollo), garantizando la encriptación en tránsito para todas las comunicaciones y protegiendo los datos.

Tabla 3:

Descripción de red y puertos

Puerto	Servicio	Host Alojado	Protocolo/Estado	Propósito
3000	Node.js API	192.168.56.20	HTTPS	Acceso principal a la API desde el frontend.
5432	PostgreSQL	192.168.56.10	Interno	Conexión exclusiva para la API (VM 2) a la Base de Datos.
443	HTTPS	Localhost	Desarrollo	Utilizado para pruebas de desarrollo local y acceso a la documentación.

Tecnologías de la infraestructura

El software clave de la infraestructura es el siguiente:

- Servidor de Aplicaciones: Node.js ejecutando el framework Express.js.
- Servidor de Base de Datos: PostgreSQL.

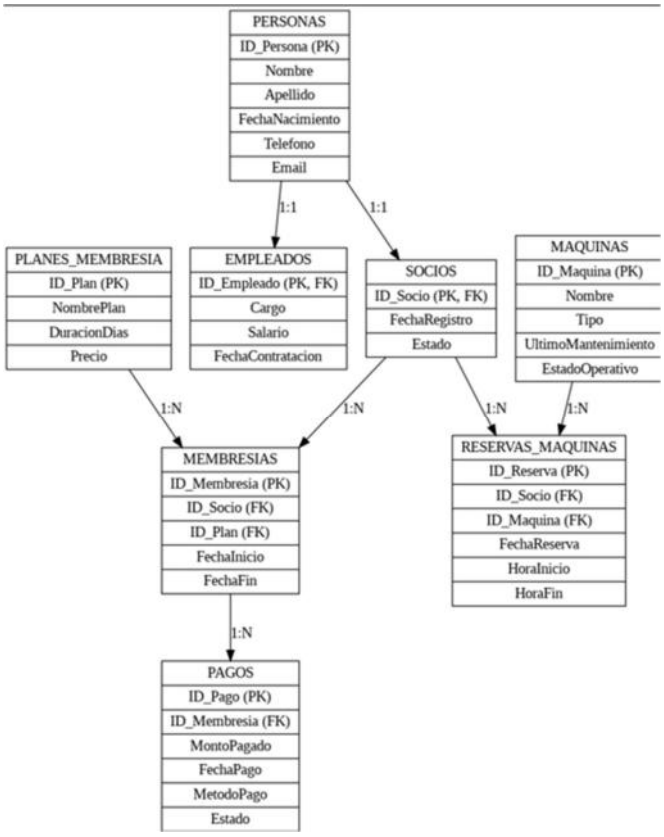
- Comunicación: HTTPS (para el tráfico de la API) y TCP (para la conexión interna a la BDD).
- Swagger/OpenAPI para la documentación de la API

3.3. Diseño de la base de datos

La capa de datos del sistema informático utiliza PostreSQL como motor de base de datos se encuentra alojada en VM PosgreSQL en la 192.168.56.10 en el puerto 5432. El diseño de la base de datos es relacional, organizada en un conjunto de tablas interconectadas para gestionar la información del gimnasio.

Figura 13:

Diagrama Entidad_Relación de la BD



Modelo de datos y tablas principales

El sistema está basado en 8 tablas principales que representan las entidades clave del modelo de negocio. Las relaciones entre las tablas se establecen mediante el uso de claves primarias PK y claves foráneas FK.

Tabla 4:

Descripción de las tablas de la base de datos

Entidad	**Clave Primaria (PK)**	**Claves Foráneas (FK)**	**Campos de Interés para Seguridad**
PERSONAS	id_persona	N/A	email, password (encriptado con crypt)
SOCIOS	id_socio	PERSONAS	Estado
EMPLEADOS	id_empleado	PERSONAS	cargo, estado
PLANES_MEMBRESIA	id_plan	N/A	nombreplan (UNIQUE)
MEMBRESIAS	id_membresia	SOCIOS, PLANES_MEMBRESIA	fecha_inicio, fecha_fin, estado

PAGOS	id_pago	MEMBRESIAS	monto_pagado, metodo_pago
MAQUINAS	id_maquina	N/A	estado
RESERVAS_MAQUINAS	id_reserva	SOCIOS, MAQUINAS	fecha_reserva, hora_inicio, hora_fin

---

Medidas de seguridad y persistencia

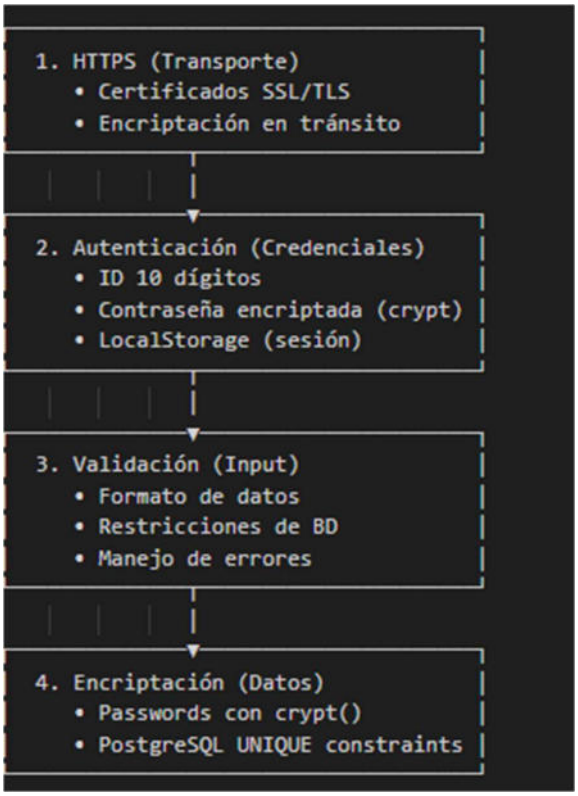
En cuanto a la seguridad de los datos y las restricciones de integridad, el campo password de la tabla personas que contiene las contraseñas de acceso al sistema, se encuentra encriptado utilizando la función crypt () de posteSQL. Además, se aplican restricciones de base de datos, como UNIQUE constrains e indices para asegurar la calidad de los datos y evitar la duplicación no deseada de los mismos.

Figura 14:

Restricciones de seguridad

Figura 15:

Capas de seguridad



Interacción con la API

La capa de persistencia juega un papel importante en la gestión de los flujos del negocio. En la Autenticación, la Base de Datos (BDD) es consultada mediante una operación SELECT en la tabla PERSONAS para validar las credenciales proporcionadas y recuperar el rol del usuario.

En el proceso de la compra de una Membresía, se ejecutan Transacciones complejas que involucran múltiples operaciones INSERT y SELECT coordinadas entre las tablas MEMBRESIAS y PAGOS, lo que asegura la atomicidad de la operación, garantizando que esta se complete totalmente o no se realice en absoluto.

Figura 16:

Flujo de consulta de datos

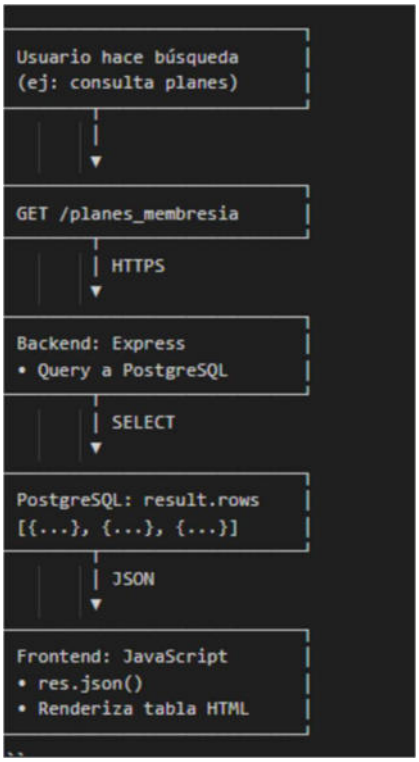


Figura 17:

Estructura de Request y Response

```javascript
{
  method: 'POST|GET|PUT|DELETE',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    // Datos a procesar
  })
}
```

Response Success (200/201):

```javascript
{
  "mensaje": "Operación exitosa",
  "data": { /* datos */ }
}
```

Response Error (400/404/500):

```javascript
{
  "mensaje": "Descripción del error"
}
```

3.4. Desarrollo de la API de gestión del gimnasio

La capa de aplicación de nuestro sistema informático es una API Restful que concentra la lógica del negocio y sirve como gateway entre los clientes (frontend) y la Base de Datos. La API fue desarrollada utilizando JavaScript por parte del servidor, utilizando Node.js y el framework Express.js.

La parte del servicio se encuentra alojada en la VM Windows Service con dirección ip 192.168.56.20 y se encuentra configurada para escuchar en el puerto 3000 y utiliza el protocolo HTTPS para garantizar la encriptación de todas las comunicaciones en tránsito. Su conectividad con la Base de Datos PostgreSQL se gestiona a través del archivo db.js, que es responsable de administrar el pool de conexión.

Figura 18:

Pool de conexión a la base de datos

3.5 Estructura y definición de los Endpoints

La API cuenta con más de 50 endpoints permitiendo la gestión de todas las entidades del gimnasio. La estructura sigue una organización por módulos de negocio.

Tabla 5:

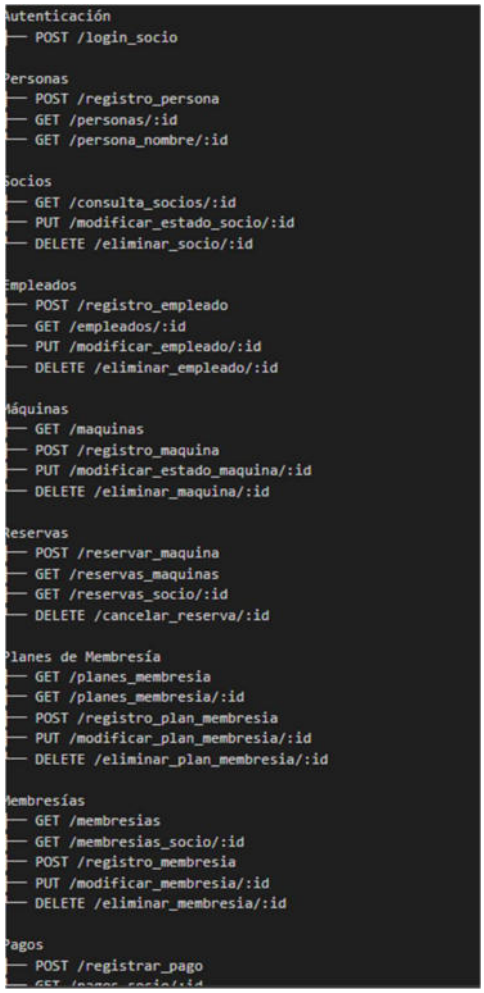
Definición de los endpoints

Módulo	**Funcionalidades (Ejemplos)**	**Métodos HTTP Clave**
Autenticación	Inicio de sesión de socios y empleados	POST /login_socio
Personas/Socios	Registro, consulta y gestión de estado de miembros	POST /registro_persona, GET /personas/:id, PUT /modificar_estado_socio/:id

Recursos	Gestión de máquinas e inventario	GET /maquinas, POST /registro_maquina
Transacciones	Proceso de alta de membresías y registro de pagos	POST /registro_membresia, POST /registrar_pago
Reservas	Gestión de horarios de máquinas	POST /reservar_maquina, DELETE /cancelar_reserva/:id

Figura 19:

Estructura de los endpoints



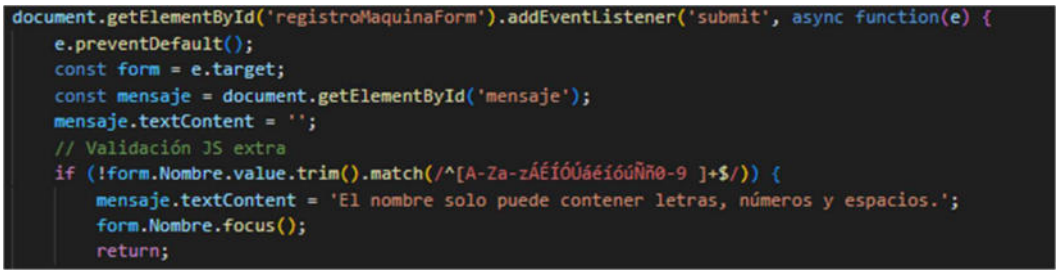
La API utiliza un protocolo de comunicación basado en **JSON**, tanto para el cuerpo de las solicitudes como para el formato de las respuestas (*Response*), utilizando códigos de estado HTTP estándar (200, 201, 400, 404, etc.).

Validación de datos

La API de Express.js tiene la responsabilidad de garantizar la calidad y robustez de las interacciones que ocurren en el sistema, lo cual se consigue mediante la validación exhaustiva del formato de los datos que se ingresan en los endpoints. Adicionalmente, implementa un sofisticado manejo de errores que no solo previene caídas del servidor, sino que también responde a los clientes con mensajes descriptivos y estandarizados en formato JSON.

Figura 20

Validación del ingreso de datos.



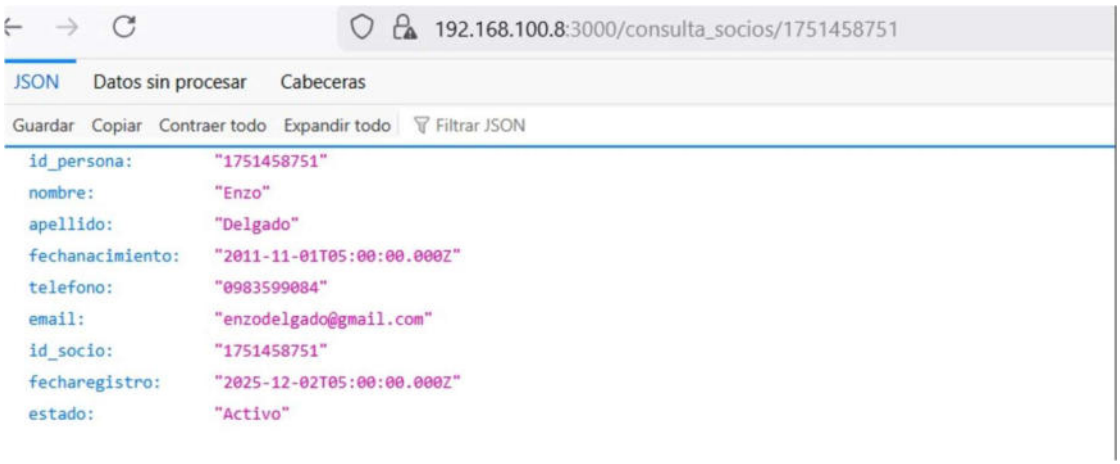
```
document.getElementById('registroMaquinaForm').addEventListener('submit', async function(e) {
    e.preventDefault();
    const form = e.target;
    const mensaje = document.getElementById('mensaje');
    mensaje.textContent = '';
    // Validación JS extra
    if (!form.Nombre.value.trim().match(/^[A-Za-zÁÉÍÓÚáéíóúÑñ0-9 ]+$/)) {
        mensaje.textContent = 'El nombre solo puede contener letras, números y espacios.';
        form.Nombre.focus();
        return;
```

Control de accesos (CORS)

La política de CORS se encuentra habilitada, lo que permite o restringe que dominios puedan realizar peticiones al servidor, factor importante en la seguridad de aplicativos webs.

Figura 20:

Política CORS



Documentación

Se utilizó Swagger/OpenApi para documentar todos los endpoints en la ruta /api-docs.Esto es fundamental al momento de realizar el analsis ya que proporciona una descripción completa de la superficie de ataque.

Figura 21:

Documentación Swagger

CAPITULO 4

4. METODOLOGÍA

4.1. Enfoque de la Investigación

El enfoque metodológico que se utilizará para la ejecución de pruebas de seguridad de API REST, tomando como base **OWASP API Security Top 10**, se centrará en una evaluación exhaustiva y estructurada de la superficie de ataque de la interfaz.

Para lo cual se ejecutarán pruebas de **Caja Gris (*Grey Box Testing*)**, ya que combina la visibilidad interna (documentación, credenciales) con la perspectiva de ataque externo, optimizando los resultados de las pruebas.

El proceso se dividirá en cuatro fases principales, para que el análisis sea completo y alineado con los riesgos más críticos: Reconocimiento, Detección de Vulnerabilidades, Explotación y Reporte

FASE I. Reconocimiento

En esta fase se recolectará información que permita comprender la funcionalidad, los *endpoints* y la estructura de datos de la API. Su objetivo es crear un mapa completo de la superficie de ataque, incluyendo tanto los *endpoints* documentados como los ocultos, antes de iniciar el ataque real.

FASE II. Detección de vulnerabilidades

En esta fase, se ejecutarán casos de prueba, enfocados en las 10 categorías de riesgo de OWASP.

Tabla 5:

Estrategias de pruebas para API Top 10

OWASP API Top 10	Estrategia de Prueba
API1: BOLA (Autorización Rota a Nivel de Objeto)	Intentar modificar el ID de un recurso en rutas, parámetros o *payloads* para acceder a datos de otros usuarios sin cambiar el token de sesión.
API2: Broken Authentication (Autenticación Rota)	Probar *endpoints* de autenticación sin *rate limiting* para ataques de fuerza bruta. Analizar la complejidad y el tiempo de vida de los tokens JWT/Cookies.
API3: BOPLA (Autorización Rota a Nivel de Propiedad)	Añadir o modificar campos de solo lectura (ej. isAdmin: true, user_role: admin) en peticiones de creación o actualización de recursos (*Mass Assignment*).
API4: Unrestricted Resource Consumption (Consumo Ilimitado)	Enviar *payloads* masivos (XML/JSON), intentar consultas con filtros complejos o consumir archivos grandes para provocar fallos o latencia en el servidor (DoS).
API5: BFLA (Autorización Rota a Nivel de Función)	Usar tokens de roles inferiores (ej. Usuario Básico) para acceder a funciones o *endpoints* destinados a roles superiores (ej. Administrador).

API6: Unrestricted Access to Sensitive Business Flows	Automatizar flujos críticos, saltar pasos, o enviar datos inválidos a un *endpoint* en un orden incorrecto para inducir estados inconsistentes.
API7: SSRF (Falsificación de Solicitud del Lado del Servidor)	Inyectar URLs que apunten a recursos internos (ej. 127.0.0.1, file:///) en parámetros que procesan URLs de entrada.
API8: Security Misconfiguration (Mala Configuración)	Escanear *headers* HTTP (CORS, HSTS, X-Powered-By), verificar la exposición de *stack traces* o mensajes de error detallados al usuario final.
API9: Improper Inventory Management (Gestión de Inventario)	Probar *endpoints* documentados de versiones anteriores (/v1) o intentar acceder a *endpoints* que se presume existen (/admin/backup).
API10: Unsafe Consumption of APIs (Consumo Inseguro de APIs)	Si la API consume servicios de terceros, inyectar código malicioso en la respuesta simulada de ese tercero para ver si el *backend* lo procesa sin validar.

FASE III. Reporte

Se evalúa el impacto de cada vulnerabilidad y se genera la documentación final.

1. **Clasificación de Severidad:** Todas las vulnerabilidades se clasifican utilizando el marco **CVSS** (Common Vulnerability Scoring System) o un sistema de riesgo (Crítico, Alto, Medio, Bajo).

2. **Recomendaciones de Mitigación:** Proporcionar soluciones técnicas detalladas para el equipo de desarrollo, alineadas con las mejores prácticas de seguridad.

3. **Generación de Reporte:** Documentar los hallazgos, incluyendo la descripción del riesgo, los pasos de reproducción y la evidencia (peticiones y respuestas capturadas).

4.2. Herramientas de Análisis

Las herramientas utilizadas para el desarrollo de las pruebas se seguridad del API fueron: Burp Suite y Owas zap, a continuación, la descripción de cada una de ellas:

4.2.1. Burpsuite

Es una plataforma de seguridad desarrollada por PortSwigger que permite realizar pruebas de penetración en aplicaciones web y descubrir sus vulnerabilidades.

Esta suite se sitúa entre el navegador del usuario y la aplicación web objetivo, actuando como un proxy que intercepta y analiza el tráfico HTTP/S, lo que permite: inspeccionar, modificar y reproducir las interacciones entre el cliente y el servidor para identificar posibles vulnerabilidades.

Características Principales de Burp Suite

Burp Suite ofrece una amplia gama de herramientas y funcionalidades que facilitan el análisis y la evaluación de la seguridad en aplicaciones web. A continuación, se detallan algunas de ellas:

1. Proxy Interceptador

Actúa como un proxy entre el navegador y el servidor web, permitiendo interceptar, inspeccionar y modificar las solicitudes y respuestas HTTP/S en tiempo real. Esta funcionalidad es esencial para analizar el tráfico y detectar posibles vulnerabilidades.

2. Escáner de Vulnerabilidades

Automatiza la detección de vulnerabilidades comunes en aplicaciones web, como **inyecciones SQL**, **cross-site scripting (XSS)** y **configuraciones de seguridad deficientes**. Proporciona informes detallados con recomendaciones para mitigar los riesgos identificados.

3. Intruder

Permite realizar ataques automatizados y personalizados contra aplicaciones web, como **fuerza bruta**, **enumeración de parámetros** y **pruebas de inyección**. Es altamente configurable y puede adaptarse a diversos escenarios de prueba.

4. Repeater

Facilita la repetición manual de solicitudes HTTP/S con modificaciones específicas para probar cómo responde la aplicación a diferentes entradas. Es útil para realizar pruebas manuales de vulnerabilidades identificadas.

5. Sequencer

Analiza la aleatoriedad de tokens y otros datos generados por la aplicación para evaluar la solidez de los mecanismos de seguridad implementados, como los identificadores de sesión.

6. Decoder

Permite codificar y decodificar datos en múltiples formatos, facilitando el análisis de información cifrada o codificada que se encuentra en las solicitudes y respuestas.

7. Comparer

Herramienta que compara dos conjuntos de datos para identificar diferencias, útil para analizar cómo cambian las respuestas de la aplicación ante diferentes solicitudes.

8. Extender

Ofrece la posibilidad de ampliar las funcionalidades de Burp Suite mediante la instalación de extensiones desarrolladas por terceros o personalizadas, adaptando la herramienta a necesidades específicas.

Versiones de Burp Suite

A continuación, se describen las versiones disponibles de Burp Suite:

1. Community Edition

Versión gratuita que incluye herramientas básicas como el **Proxy**, **Repeater**, **Decoder** y **Comparer**. Es adecuada para quienes se inician en pruebas de seguridad o para proyectos con recursos limitados.

2. Professional Edition

Versión de pago que ofrece funcionalidades avanzadas, incluyendo el **Escáner de Vulnerabilidades**, **Intruder** mejorado y acceso a actualizaciones y soporte técnico. Es ideal para profesionales de la seguridad que requieren herramientas más potentes y completas.

3. Enterprise Edition

Diseñada para organizaciones que necesitan realizar pruebas de seguridad a gran escala de forma automatizada. Incluye capacidades de integración con otros sistemas y permite programar y gestionar escaneos de manera eficiente.

Cómo Utilizar Burp Suite

A continuación, se presenta una guía básica para comenzar a utilizar **Burp Suite** en la evaluación de la seguridad de aplicaciones web:

1. Configuración del Proxy

- **Instalación**: Descargar e instalar Burp Suite desde el sitio oficial de PortSwigger.
- **Configuración del Navegador**: Configurar el navegador para que utilice el proxy de Burp Suite, permitiendo interceptar el tráfico entre el navegador y la aplicación web.
- **Certificado SSL/TLS**: Importar y confiar en el certificado CA de Burp Suite en el navegador para interceptar tráfico HTTPS sin generar alertas de seguridad.

2. Intercepción y Modificación de Tráfico

- **Interceptar Solicitudes**: Activar el modo de intercepción para capturar solicitudes HTTP/S enviadas por el navegador.
- **Modificar Solicitudes**: Editar las solicitudes interceptadas para probar cómo responde la aplicación a diferentes entradas, identificando posibles vulnerabilidades.

3. Escaneo de Vulnerabilidades

- **Configuración del Escáner**: Definir el alcance y las configuraciones del escaneo según los objetivos de la prueba.

- **Ejecución del Escaneo**: Iniciar el escaneo automático para identificar vulnerabilidades en la aplicación.

- **Análisis de Resultados**: Revisar los informes generados, que incluyen detalles de las vulnerabilidades encontradas y recomendaciones para su mitigación.

4. Uso de Intruder para Ataques Personalizados

- **Configuración de Ataques**: Seleccionar la solicitud que se desea atacar y definir los parámetros que se modificarán.

- **Selección de Payloads**: Especificar los datos que se enviarán en cada solicitud para probar diferentes vectores de ataque.

- **Ejecución y Análisis**: Lanzar el ataque y analizar las respuestas de la aplicación para identificar posibles puntos débiles.

Mejores Prácticas al Usar Burp Suite

Para maximizar la eficacia y seguridad al utilizar **Burp Suite**, considerar las siguientes recomendaciones:

- **Autorización**: Asegurarse de tener permiso explícito para probar la seguridad de una aplicación antes de realizar cualquier prueba.

- **Alcance Definido**: Establecer claramente el alcance de las pruebas para evitar afectar sistemas no autorizados o causar interrupciones no deseadas.

- **Entorno Controlado**: Realizar pruebas en entornos de desarrollo o pruebas que no afecten a usuarios finales o datos sensibles.

- **Actualización Continua**: Mantener Burp Suite actualizado para aprovechar las últimas mejoras y correcciones de seguridad.

4.2.2. OWASP ZAP

Zed Attack Proxy (ZAP) es una herramienta gratuita de prueba de penetración de código abierto que se mantiene bajo el paraguas del Open Web Application Security Project (OWASP). Esta herramienta está diseñada específicamente para probar aplicaciones web.

Este programa es mantenido activamente por una comunidad internacional de voluntarios, los cuales trabajan para ir mejorando la herramienta poco a poco y también incorporando nuevas características.

ZAP es lo que se conoce como un "proxy de intermediario". Se encuentra entre el navegador del probador y la aplicación web para que pueda interceptar e inspeccionar los mensajes enviados entre el navegador y la aplicación web, modificar el contenido si es necesario y reenviar esos paquetes al destino.

Figura 22:

Ubicación de Owasp zap entre el browser y la aplicación

Ventajas de ZAP:

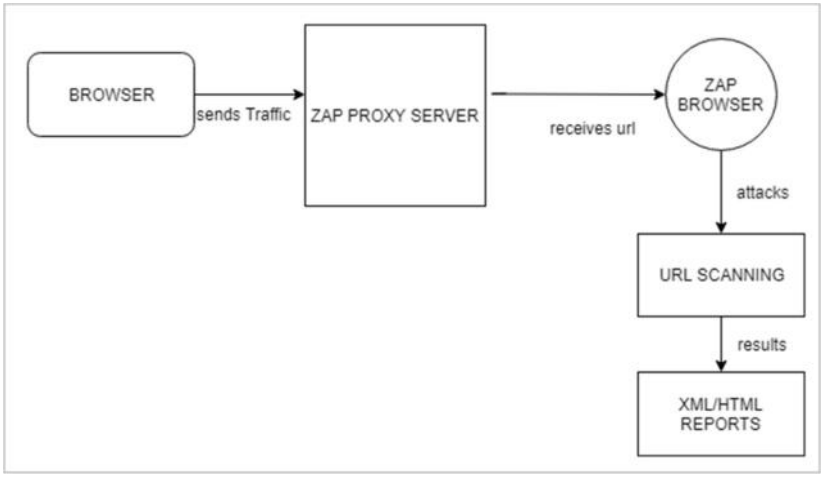
- ZAP proporciona multiplataforma, es decir, funciona en todos los sistemas operativos (Linux, Mac, Windows)
- ZAP es reutilizable
- Puede generar informes
- Ideal para principiantes
- Herramienta gratuita

Funcionamiento de la herramienta:

En esta figura se muestra de una forma un poco más extensa el funcionamiento general de la herramienta.

Figura 23:

Flujo Owasp zap



La función principal de ZAP s descubrir posibles brechas de seguridad, en programas/aplicaciones de internet a través del test de penetración (pentest), que abarcan desde el escaneo automatizado, supervisión de intrusión en modo manual, el procedimiento conocido como fuzzing, a la realización de scripting y más allá.

Principales características de ZAP

A continuación, se describen las principales funciones de OWASP ZAP:

1. **Escaneo automatizado**: Sin la necesidad de intervención por parte del operador, puede detectar problemas de seguridad frecuentes, como pueden ser la Inyección SQL, el Cross-Site Scripting (XSS) y el Cross-Site Request Forgery (CSRF) en la creación de páginas web.

2. **Supervisión manual de intrusión**: Adicional a la función de escaneo automatizado, ZAP otorga a los operadores la alternativa de ejecutar las pruebas de manera manual para detectar posibles debilidades que la versión automatizada no alcanza a detectar.

3. **Fuzzing**: ZAP permite activar la opción de fuzzing, la cual promueve el envío de información al azar a programas de internet para detectar posibles brechas de seguridad.

4. **Scripting**: Posibilidad de crear y ejecutar scripts personalizados para llevar a cabo labores específicas, como automatizar ciertas pruebas o controlar las peticiones y respuestas.

5. **Intercepción de peticiones y respuestas**: ZAP hace posible el análisis y la alteración de las peticiones y respuestas que se generan entre el usuario y el servidor, para encontrar y manejar posibles oportunidades de mejora que no serían visibles a simple vista.

Funcionamiento de ZAP

ZAP opera emulando un intermediario entre el usuario y la web aplicación que se está examinando. Todas las peticiones y respuestas, mientras el usuario opera con la web aplicación citada, se direccionan a través de ZAP. ZAP tiene la función de registrar cada interacción y luego examinarlas para localizar posibles fallos de seguridad.

ZAP tiene dos modos de operación: pasivo y activo. En el modo pasivo, ZAP se limita a registrar y examinar el tráfico que fluye entre la web aplicación y el usuario sin alterar las peticiones y las respuestas. Este modo resulta útil para detectar fallos de seguridad de forma evidente pero no puede localizar problemas más sutiles que requieren de una interacción más directa.

Por otro lado, en su modo activo, ZAP se implica en el proceso de forma más proactiva modificando las peticiones y respuestas para comprobar si la aplicación web presenta vulnerabilidades, esto puede implicar inyectar códigos dañinos en las peticiones o cambiar las respuestas para ver cómo reacciona la aplicación.

Elementos esenciales de ZAP

ZAP integra varios componentes fundamentales que cooperan entre sí para proporcionar sus habilidades de análisis de seguridad. Estos son:

1. **Proxy HTTP**: El elemento principal que registra y altera el tráfico online entre la web application y el usuario.

2. **Araña web (Spider)**: Este elemento explora la aplicación web para encontrar contenido y funcionalidades ocultas.

3. **Rastreador activo (Active Scanner)**: Este elemento realiza pruebas activas para detectar vulnerabilidades.

4. **Generador de datos aleatorios (Fuzzer)**: Este componente envía grandes cantidades de información inesperada para ver cómo reacciona la aplicación.

5. **Intentador (Forcer)**: Este componente prueba a adivinar contraseñas y otros datos confidenciales.

4.3. Fases del Análisis

4.3.1. FASE I. Reconocimiento

Mapeo Basado en la Documentación

A continuación, se detalla cómo ejecutar esta fase para una API RESTful, con enfoque en las vulnerabilidades de OWASP.

El primer paso es utilizar la información proporcionada por el equipo de desarrollo para construir la base del mapa de la API.

Análisis de la Especificación de la API

- **Recopilación:** Obtener el archivo de especificación de la API.
- **Listado Exhaustivo:** Extraer la lista completa de *endpoints* (rutas), métodos HTTP (GET, POST, PUT, DELETE) y códigos de estado de respuesta esperados.
- **Identificación de Parámetros:** Documentar los parámetros de entrada requeridos y opcionales en la ruta, la *query* (URL) y el cuerpo (*payload*).

Perfilación de Roles y Autorización (API5)

- **Definición de Roles:** Identificar y documentar todos los roles de usuario que interactúan con la API (ej. public, standard, premium, admin).

- **Matriz de Acceso:** Crear una matriz de autorización básica con los roles que pueden acceder a los endpoints

Descubrimiento Dinámico de Endpoints Ocultos

Se enfoca en encontrar endpoints que existen en producción, pero que están obsoletos, mal documentados o diseñados para uso interno.

Proxy de Intercepción y Tráfico en Vivo

- **Configuración del Proxy:** Utilizando las herramientas proxy de intercepción seleccionadas (como **Burp Suite** o **OWASP ZAP**) capturar todo el tráfico HTTP/HTTPS entre la aplicación cliente (web, móvil, escritorio) y la API.

- **Interacción Completa:** Interactuar con la aplicación cliente, ejecutando los flujos de negocio posibles (login, registro, edición de perfil, consulta, eliminación, etc.).

- **Monitoreo del Tráfico:** Análisis del log del proxy. Cualquier solicitud a la API que no esté en la documentación se añade al mapa de endpoints.

Enumeración de Directorios

- **Ataque de Diccionario:** Utilizando herramientas de *fuzzing* contra el dominio de la API. y utilizando diccionarios que contienen nombres de archivos comunes, nombres de endpoints por defecto (/admin, /status, /backup) y métodos de API conocidos.

- **Identificación de Versiones Obsoletas:** Búsqueda de versiones descontinuadas (ej. intentar acceder a /v1/users si la documentación solo muestra

/v3/users). Estos endpoints antiguos a menudo carecen de los parches de seguridad de las versiones actuales.

Mapeo de Parámetros y Objetos de Datos (API3)

- **Campos de Entrada:** Para cada método POST y PUT, identificar todos los campos aceptados.

- **Campos de Salida:** Para cada método GET, documentar todos los campos devueltos en la respuesta. Esto es crucial para identificar la **Exposición Excesiva de Datos** (ej. si la respuesta incluye *hashes* de contraseñas, claves internas o información de usuario no solicitada).

Descubrimiento de IDs y Relaciones (API1)

- **Identificación de IDs:** Documentar todos los identificadores utilizados en las rutas (ej. user_id, order_id, client_id).

- **Prueba de Relaciones:** Determinar si los IDs de un usuario estándar son secuenciales o predecibles.

El resultado final de esta fase es un **Mapa de la API** exhaustivo que lista todos los *endpoints*, los métodos, los parámetros esperados/observados y las relaciones de autorización, con lo cual se ha recopilado toda la información necesaria para las pruebasdinámicas.

A continuación, se describe el cuadro en el que documentará el Mapa de la API:

Tabla 6:

Mapa de la API

Endpoint / Método	**Parámetros (Path/Query/Body)**	**Roles Permitidos (API5)**	**Observaciones de Seguridad (API3/API9)**	**Resultado del Descubrimiento Dinámico**
GET /api/v3/user/{id}	**Path:** user_id (numérico) **Query:** expand (opcional)	**Standard**, Premium, Admin	**Salida:** Expone internal_client_id, password_hash. **Riesgo:** Exposición excesiva de datos (**API3**).	GET /v1/user/10 está activo y devuelve datos. (**API9** - Versión obsoleta).
POST /api/v3/order	**Body:** product_id, quantity, shipping_address, price (cliente-controlado)	**Standard**, Premium	**Entrada:** Acepta price en el body. **Riesgo:** Posible *Mass Assignment* o manipulación de precio.	Se observó que POST /v2/order/create está activo, pero no documentado.

PUT /api/v3/profile/edit	**Body:** name, email, preferences, role (no esperado)	**Standard**, Premium	**Entrada:** Se acepta un campo role en la carga útil que es ignorado por la documentación. **Riesgo:** Posible *Mass Assignment* (**API3**).	El tráfico en vivo reveló el parámetro de *query* ?lang=es.
GET /api/v3/orders	**Query:** limit, offset	**Standard**, Premium, Admin	**Control:** Los parámetros limit y offset no tienen un máximo definido en la especificación. **Riesgo:** Consumo de recursos (**API4** - Pendiente de prueba).	El endpoint /admin/status está activo y no requiere autenticación. (**API9**).

DELETE /api/v3/user/{id}	**Path:** user_id (numérico)	**Admin**	**Acceso:** Solo rol Admin, crítico. **Riesgo:** BFLA si es accesible por otro rol (**API5**).	El user_id en el tráfico en vivo es secuencial. **Riesgo:** IDs predecibles (**API1**).

4.3.2. FASE II Detección de Vulnerabilidades

A continuación, se detallará como ejecutar cada prueba utilizando las dos herramientas seleccionadas para el presente análisis:

4.3.2.1. API1 BOLA

Objetivo

Acceder, modificar o eliminar un recurso (ej. máquina, reserva, usuario) que pertenece al Usuario B mientras se está autenticado únicamente como Usuario A.

Prerrequisitos

Dos Cuentas de Prueba: Se necesitará dos usuarios: Usuario A (el atacante) y Usuario B (la víctima).

ID de Recurso: Conocer un ID de recurso que pertenece al Usuario B (ej. order_id: 500).

Paso a Paso para la Explotación con Burp Suite (Repeater)

1. **Capturar solicitud legítima:** Asegurarse de que el Proxy de Burp esté interceptando. Autentícarse como el Usuario A y realiza una solicitud para acceder a su propio recurso (GET /api/v1/orders/101).

2. **Enviar a Repeater:** En la pestaña Proxy - Intercept, dar clic derecho sobre la solicitud y seleccionar "Send to Repeater". Desactivar la intercepción.

3. **Preparar ataque:** Ir a la pestaña Repeater e identificar el ID del Objeto (101) en la URL o el cuerpo.

4. **Ejecutar la prueba (Manipulación):** Cambiar el ID de objeto por uno que pertenece al Usuario B (202).

5. **Verificar análisis:** Enviar la solicitud. Si el servidor devuelve 200 OK con los datos del Usuario B (a pesar de usar el token del Usuario A), la API es Vulnerable (BOLA). El resultado seguro es un error de autorización (401 o 403).

4.3.2.2. API2 Broken Authentication

Objetivo

Comprobar si el endpoint de autenticación (/auth/login) tiene mecanismos de defensa efectivos (Rate Limiting o Throttling) que impidan un ataque automatizado de fuerza bruta.

Prerrequisitos

1. **Burp Suite:** Instalado y configurado como proxy en el navegador/cliente.

2. **Credenciales de Prueba:** Un nombre de usuario válido para el cual se intentará adivinar la contraseña (admin).

3. **Lista de Contraseñas:** Un archivo de texto (wordlist) con contraseñas comunes o de diccionario.

Paso a Paso para la Explotación con Burp Suite

Escenario: Ataque de Fuerza Bruta (Password Spraying)

Paso 1: Interceptar la Solicitud de Login

1. Activar la función Intercept de Burp Suite (pestaña Proxy).

2. En el cliente o navegador, intentar iniciar sesión con el usuario de prueba (admin) y una contraseña incorrecta (password).

3. Burp Suite capturará la solicitud POST /auth/login.

Paso 2: Enviar al Módulo de Ataque

1. En la ventana de Proxy de Burp, hacer clic derecho sobre la solicitud capturada.

2. Seleccionar "Send to Intruder".

3. Asegurarse de desactivar la función Intercept en el Proxy para no interferir con la prueba.

Paso 3: Configurar el Ataque en Intruder

1. Ir a la pestaña **Intruder**.

2. En la subpestaña **Positions**:

a. Hacer clic en Clear para eliminar todas las posiciones predeterminadas de ataque.

b. Seleccionar y marcar solo el valor de la contraseña dentro del cuerpo del payload (o en el campo de la URL, si la API usa query parameters). Hacer clic en Add para definir esta área como la de ataque.

c. Verificar que el Attack Type esté configurado como Sniper (para atacar solo el campo de la contraseña).

3. En la subpestaña Payloads:

a. En el campo Payload Options, hacer clic en Load y selecciona el archivo de texto (wordlist) con las contraseñas que se van a probar.

4. En la subpestaña Options (Opcional, pero recomendado):

a. Ir a la sección Grep - Extract para configurar extractores que automáticamente busquen mensajes de error/éxito en la respuesta ("Login Exitoso" o "Contraseña Incorrecta").

Paso 4: Ejecutar el Ataque y Análisis

1. Hacer clic en "Start attack" (normalmente arriba a la derecha). Burp Suite comenzará a enviar peticiones rápidamente, probando cada contraseña de la lista.

2. Análisis de Resultados (Vulnerabilidad):

a. **Vulnerable (Ausencia de Rate Limiting):** Si el ataque se completa rápidamente y todos los códigos de estado son 200 OK o 401 Unauthorized por

contraseña incorrecta, y el tiempo de respuesta es constante. Esto significa que no hay defensa contra la automatización.

b. **Vulnerable (Enumeración de Usuarios):** Si al probar contraseñas incorrectas la respuesta cambia significativamente cuando el usuario es válido (un delay de 5 segundos) frente a un usuario inválido (respuesta instantánea). Esto permite a un atacante validar si un nombre de usuario existe.

c. **Seguro:** Si, después de un número bajo de peticiones (5 o 10), el servidor comienza a devolver códigos de estado 429 Too Many Requests, o el tiempo de respuesta de cada petición aumenta significativamente (throttling).

Escenario 2: Prueba de Debilidad en el Token JWT

El objetivo es validar que el token de sesión (JWT) no pueda ser alterado fácilmente para escalar privilegios.

Paso A: Obtener el Token

1. Iniciar sesión con un usuario estándar y capturar el token JWT de la respuesta del login.

Paso B: Manipular el Token (JWE/JWS)

1. Utilizar la extensión JWT Editor (si usas Burp Pro) o una herramienta en línea para decodificar la parte del Payload del token JWT (la sección central).

2. Ataque de Rol (BOLA/BFLA): Modificar el claim de rol de {"role": "user"} a {"role": "admin"}.

3. **Ataque de Firma (alg:none):** Si la API no valida la firma, cambiar el algoritmo de firma a {"alg": "none"} y eliminar la parte de la firma del token.

Paso C: Probar el Acceso Elevado

1. Cargar el token modificado en el *header* Authorization de una solicitud para un *endpoint* restringido (ej. /admin/dashboard).

2. **Vulnerable:** Si la API acepta el token modificado y otorga acceso al recurso restringido, confirma que hay fallas en la **validación de la firma** del token.

4.3.2.3. API3 BOPLA

Objetivo

Comprobar si el servidor permite a un usuario modificar propiedades de objetos que no deberían ser modificables por ese rol o nivel de permiso (un usuario estándar modificando su propio rol, balance o estado de cuenta). Esto se conoce como Mass Assignment o Asignación Masiva.

Prerrequisitos

1. **Burp Suite:** Instalado, configurado y activo como *proxy* de intercepción.

2. **Una Cuenta de Prueba:** Un **Usuario A** con permisos estándar (no administrativo).

3. **Campo Sensible:** Conocer el nombre de un campo sensible en la base de datos o modelo de datos al que el usuario no debería tener acceso (role: admin, accountStatus: approved, balance: 99999).

Paso a Paso para la Explotación con Burp Suite (Repeater)

El ataque se centra en la manipulación del cuerpo de la petición (payload) durante una operación de actualización (PUT o POST).

Paso 1: Interceptar una Petición de Actualización Válida

1. **Inicio de Sesión:** En el navegador, iniciar sesión con el **Usuario A** (el usuario de bajo privilegio).

2. **Actualización Normal:** Realiza una acción de actualización de perfil o de un recurso permitido (cambiar tu nombre o dirección).

3. **Captura la Solicitud:** Burp Suite capturará la solicitud PUT o POST del *endpoint* de actualización.

Ejemplo de Solicitud Normal (Usuario A):

```
HTTP

PUT /api/v1/profile/self HTTP/1.1
Authorization: Bearer [TOKEN_DEL_USUARIO_A]

{"name": "Juan", "address": "Calle Falsa 123"} <-- Campos permitidos
```

4. Hacer clic derecho sobre la solicitud capturada y seleccionar **"Send to Repeater"**.

Paso 2: Inyectar Propiedades Prohibidas (Mass Assignment)

1. Ir a la pestaña **Repeater**.

2. En el cuerpo de la petición (payload), **añadir uno o más campos sensibles/privilegiados** que se hayan identificado y que el Usuario A no debería poder modificar.

Solicitud Maliciosa (Inyección de Propiedad):

```
HTTP

PUT /api/v1/profile/self HTTP/1.1
Authorization: Bearer [TOKEN_DEL_USUARIO_A]
{

"name": "Juan",

"address": "Calle Falsa 123",

"role": "admin", <-- CAMPO INYECTADO (Ataque BOPLA)

"accountStatus": "approved" <-- CAMPO INYECTADO (Ataque BOPLA)

}
```

3. Asegurarse de que el Token de Autorización sigue siendo el del Usuario A.

Paso 3: Ejecutar la Petición y Analizar la Respuesta

1. Hacer clic en **"Send"** en la pestaña **Repeater**.

2. Analizar el código de estado y el cuerpo de la respuesta.

3. **Verificación Post-Ataque:** Confirmar si la base de datos realmente actualizó los campos inyectados. Intentar acceder al perfil del Usuario A para ver si su rol se convirtió en "admin".

4. **Verificar Análisis:** Si el servidor devuelve 200 OK y el campo de privilegio (role: admin) fue aplicado, la API es **Vulnerable (BOPLA)**. El resultado seguro es que el servidor ignore el campo o rechace la solicitud (400 o 403).

4.3.2.4. API4 Consumo Ilimitado

Objetivo

Verificar la presencia y efectividad de los mecanismos de **Rate Limiting**, **Throttling** y **validación de límites de tamaño (*payload*)** para prevenir la degradación del servicio o el fallo total debido a peticiones excesivas o sobredimensionadas.

Prerrequisitos

1. **Burp Suite Professional (o ZAP):** La edición profesional es preferible para *rate limiting* por su velocidad y estabilidad.

2. **Dos Endpoints de Prueba:**

    a. Un *endpoint* **simple y de bajo consumo**

    (ej. `GET /api/v1/status`).

    b. Un *endpoint* **complejo y de alto consumo**

    (ej. `GET /api/v1/reports?filter=complex_query` o un `POST` con *`payload`* `grande`).

Paso a Paso para la Explotación con Burp Suite (Intruder)

Escenario 1: Ataque de Rate Limiting (Peticiones Excesivas)

El objetivo es saturar el servidor con peticiones al *endpoint* más costoso.

Paso 1: Interceptar y Enviar a Intruder

1. En el navegador, ejecutar una solicitud al *endpoint* de **alto consumo** (ej. la generación de un reporte).

2. Capturar la solicitud (GET o POST) en el **Proxy** de Burp.

3. Hacer clic derecho sobre la solicitud y seleccionar **"Send to Intruder"**.

Paso 2: Configurar el Ataque de Carga (Núcleo de la Prueba)

1. Ir a la pestaña **Intruder**.

2. En **Positions**: Asegurarse de que **no haya posiciones de ataque definidas**. El objetivo es enviar la *misma* petición miles de veces.

3. En **Payloads**:

    a. **Payload Type:** Seleccionar **"Null Payloads"**.

    b. **Payload Options:** Establecer el **número de peticiones** que deseas enviar (ej. 5,000 o 10,000).

4. En **Options**:

    a. **Número de Threads:** Aumentar el número de *threads* (hilos) para simular muchos usuarios simultáneos (ej. 20-50).

b. **Lanzar y Monitorear:** Ejecutar el ataque y **monitorear el tiempo de respuesta** y los códigos HTTP.

Paso 3: Análisis de Resiliencia

- **Vulnerable (DDoS/DoS Confirmado):** Si todas las peticiones regresan **200 OK** y el tiempo de respuesta promedio **aumenta drásticamente** (ej. de 50ms a 5 segundos), o si el servidor **deja de responder (5xx)**.

- **Seguro (Rate Limiting Efectivo):** Si, después de un umbral razonable (ej. 100 peticiones), el servidor comienza a devolver sistemáticamente el código **429 Too Many Requests**, o si la velocidad de ataque se detiene automáticamente.

Escenario 2: Ataque de Consumo de Memoria/CPU (Payloads Ilimitados)

El objetivo es forzar al servidor a consumir CPU o memoria al procesar un *payload* que es desproporcionadamente grande o complejo.

Paso 1: Interceptar y Enviar la Solicitud de Carga

1. Identificar un *endpoint* que acepte grandes entradas (ej. un POST de un JSON para crear un objeto complejo o una lista de ítems).

2. Capturar la solicitud POST en el Proxy y envíarla a **Intruder**.

Paso 2: Configurar el Ataque de Payload Grande

1. En **Positions**: Definir el valor del campo que contendrá los datos (ej. un *array* de ítems) como la posición de ataque.

2. En **Payloads**:

a. **Payload Type:** Seleccionar **"Numbers"** o **"Custom Iterator"** para generar datos de prueba.

b. **Alternativa de Payload:** Crear un *payload* JSON extremadamente grande (ej. 10MB con *arrays* anidados) y utilizar el **Repeater** para enviarlo de forma manual.

3. **Monitorear:** Envíar el *payload* y monitorear el tiempo de respuesta.

Paso 3: Análisis de Límite de Tamaño

- **Vulnerable (Fallo en Memoria):** Si el servidor intenta procesar el *payload* masivo, y el tiempo de respuesta se dispara o el servidor responde con un error interno (5xx), indica que no hay un límite de tamaño de *payload* configurado en el *firewall* o la capa de aplicación.

- **Seguro:** El servidor rechaza inmediatamente el *payload* con un código **413 Payload Too Large** o un error de validación de esquema, incluso antes de que el código de la aplicación lo procese.

4.3.2.5. API5 BFLA

Objetivo

Comprobar que un usuario con privilegios bajos (ej. Usuario Estándar) **no puede acceder, ejecutar o descubrir** *endpoints* destinados exclusivamente a usuarios con privilegios altos (ej. Administrador o Super-Usuario).

Prerrequisitos

1. **Dos Cuentas de Prueba:**

a. **Usuario A (Atacante):** Un usuario con **bajos privilegios** (ej. Usuario Estándar).

b. **Usuario B (Modelo):** Un usuario con **altos privilegios** (para modelar las rutas prohibidas).

2. **Endpoint Prohibido Conocido:** Conocer una ruta que es exclusiva para el rol alto (ej. `POST /api/v1/admin/purge_users`).

Paso a Paso para la Explotación con Burp Suite (Repeater)

Escenario: Acceso a Función Restringida

Paso 1: Obtener el Token de Acceso del Atacante

1. **Inicio de Sesión:** En el navegador, inicia sesión con el **Usuario A** (el usuario de **bajo privilegio**).

2. **Captura del Token:** Captura una solicitud cualquiera del Usuario A para obtener su **Token de Sesión** (JWT o Cookie).

3. Envíar la solicitud a **Repeater**.

Paso 2: Construir la Petición Prohibida

1. En la pestaña **Repeater**, modificar la URL de la solicitud para apuntar al *endpoint* prohibido que se desea probar.

a. Ejemplo (Intentando Borrar Usuarios):

i. **Método:** Cambiar a DELETE o POST.

ii. **URL:** Cambiar a la ruta administrativa (ej. `/api/v1/admin/purge_users`).

iii. **Body:** Ajustar el cuerpo si el *endpoint* requiere parámetros (ej. `{"status": "deleted"}`).

Solicitud Maliciosa (Token Estándar, Ruta Admin):

```
HTTP

DELETE /api/v1/admin/purge_users HTTP/1.1 <-- FUNCIÓN PROHIBIDA
Host: api.ejemplo.com
Authorization: Bearer [TOKEN_DEL_USUARIO_A] <-- TOKEN ESTÁNDAR
```

2. Asegurarse de que el **Token de Autorización no se ha modificado**; debe seguir siendo el del Usuario A (bajo privilegio).

Paso 3: Ejecutar la Petición y Analizar la Respuesta

1. Hacer clic en **"Send"** en la pestaña **Repeater**.

2. Analizar el código de estado HTTP y el cuerpo de la respuesta:

Vulnerable (Fallo en Memoria): La operación se ejecuta 200 OK (ej. devuelve una lista de usuarios borrados o confirma la acción). La API no verificó el rol del usuario antes de ejecutar la función.

Seguro: La API verificó correctamente el rol del Usuario A y bloqueó el acceso a la función restringida.

4.3.2.6. API6 Business Flows

Objetivo

Comprobar que los flujos de negocio sensibles (login, registro, reserva, etc) están protegidos contra la automatización, la omisión de pasos y las manipulaciones lógicas que conducen al abuso.

Prerrequisitos

1. **Flujo de Negocio Conocido:** Identificar y mapear un flujo de varios pasos
2. **Cuenta de Prueba:** Una cuenta de usuario estándar.

Paso a Paso para la Explotación con Burp Suite (Repeater & Intruder)

El ataque se divide en dos escenarios comunes: Omisión de Pasos y Abuso de Automatización.

Escenario 1: Omisión de Pasos (Skipping Steps)

El objetivo es saltarse una validación crítica dentro de una secuencia lógica.

Paso 1: Mapear la Secuencia Completa

1. En el navegador, ejecutar el flujo de negocio sensible de forma normal, asegurándote de que **Burp Suite Proxy** está activo.

2. Documentar la secuencia exacta de peticiones (URL y Método) que se envían a la API (ej. `Petición 1: /cart/add, Petición 2: /address/validate, Petición 3: /checkout/confirm`).

Paso 2: Aislar la Petición Final

1. Tomar la **petición final** del flujo (ej. `/checkout/confirm` o `/registration/final`). Esta es la petición que completa la acción.

2. Envíar esta petición final al módulo **Repeater**.

Paso 3: Manipular la Secuencia

1. Intentar ejecutar la petición final (/checkout/confirm) sin haber ejecutado las peticiones anteriores.

    a. *Opción A:* Si la API utiliza *tokens de estado* de flujo, intenta omitir la petición que genera el token del paso anterior.

    b. *Opción B:* Si la API usa un parámetro para indicar el paso (step=4), intentar enviar directamente step=4 sin haber pasado por step=1, step=2, etc.

2. Hacer clic en **"Send"** y analiza la respuesta.

    Vulnerable: Respuesta **200 OK** El servidor procesa la petición final sin validar que los pasos anteriores se completaron.

    Seguro: Respuesta **403 Forbidden / 400 Bad Request**. Mensaje de error (ej. "Secuencia de pasos incorrecta" o "Token de sesión de flujo inválido"). La API mantiene el estado del flujo y obliga a completarlo secuencialmente.

Escenario 2: Abuso de Automatización (Rate Limiting en Lógica)

El objetivo es abusar de una función con límite implícito (ej. votar, aplicar un cupón único, reservar un artículo de inventario limitado).

Paso 1: Interceptar la Función Limitada

1. Ejecutar la acción de negocio que debería estar limitada (ej. un voto en una encuesta, la aplicación de un cupón de "primer uso").

2. Capturar la solicitud (POST o GET) en el **Proxy** de Burp.

3. Envíar la solicitud al módulo **Intruder**.

Paso 2: Configurar el Ataque de Abuso

1. En **Positions de Intruder**: Asegurarse de que no haya posiciones de ataque definidas (ataque *Sniper* de un solo *payload* vacío).

2. En **Payloads**:

    a. **Payload Type:** Seleccionar **"Null Payloads"**.

    b. **Payload Options:** Configurar el número de peticiones a un número alto (ej. 1,000) para simular la automatización.

3. **Lanzar y Monitorear:** Ejecutar el ataque y monitorear la respuesta y los logs de la aplicación.

Paso 3: Análisis del Resultado Lógico

- **Vulnerable (Abuso de Inventario/Fraude):** Si el ataque resulta en 1,000 máquinas reservadas al mismo usuario. Esto significa que **la lógica de negocio no tiene un *rate limiting*** o un bloqueo de "uso único" efectivo.

- **Seguro:** La API devuelve **403 Forbidden** o un error de lógica de negocio (ej. "Cupón ya utilizado" o "Límite de votos excedido") después del primer intento o después de alcanzar el límite esperado.

4.3.2.7. API7 SSRF

Objetivo

Verificar que la API no acepta ni procesa URLs o rutas que apunten a recursos internos del servidor (ej. 127.0.0.1, archivos de sistema) o que obliguen al servidor a contactar con dominios de atacantes.

Prerrequisitos

1. **Endpoint Vulnerable:** Identificar un *endpoint* que procese una URL o ruta proporcionada por el usuario (ej. una función que descarga una imagen por URL, un *webhook*, o una función de importación).

Paso a Paso para la Explotación con Burp Suite (Repeater & Collaborator)

Escenario 1: Exfiltración de Datos de la Red Interna

El objetivo es hacer que el servidor de la API acceda a su propia red privada o a archivos de sistema sensibles.

Paso 1: Interceptar y Enviar a Repeater

1. En el navegador, ejecutar una solicitud normal que use el parámetro de URL (ej. un servicio que recibe una URL para descargar un avatar).

Ejemplo de Parámetro Válido:

```
POST /api/v1/download_avatar con payload {"url":
"https://images.ejemplo.com/avatar.jpg"}
```

2. Capturar la solicitud en el **Proxy** de Burp y envíala a **Repeater**.

Paso 2: Inyectar Direcciones Locales

1. En la pestaña **Repeater**, en el parámetro de URL, sustituir la URL externa válida por direcciones internas comunes y sensibles.

    a. **Acceso Local:** Sustituir por http://127.0.0.1/ o http://localhost/.

    b. **Acceso Interno:** Sustituir por una dirección de la red privada común (ej. http://192.168.0.1/).

    c. **Archivos de Sistema:** Sustituir por rutas de archivos locales (ej. file:///etc/passwd o file:///c:/windows/win.ini).

Solicitud Maliciosa (Ejemplo):

```
POST /api/v1/download_avatar con payload {"url":
"http://127.0.0.1/admin_status"}
```

2. Hacer clic en **"Send"** y analizar la respuesta.

Vulnerable: Respuesta **200 OK**. El cuerpo de la respuesta contiene **c**ódigo HTML o texto (ej. la página de login del servidor, una lista de archivos, o el contenido de /etc/passwd). El servidor accedió a su propia red interna y devolvió el contenido.

Seguro: Respuesta **400 Bad Request / 403 Forbidden.** La API validó la URL y bloqueó el acceso a *localhost* y esquemas de archivo.

Escenario 2: Interacción con Servicio de Atacante (Collaborator)

El objetivo es confirmar que el servidor de la API puede hacer una solicitud a un dominio externo controlado por el atacante.

Paso 1: Generar Payload en Burp Collaborator

1. Ir a la pestaña **Burp Collaborator Client**.
2. Hacer clic en **"Copy to clipboard"** para generar y copiar una URL única para esta prueba (ej. xyz.burpcollaborator.net).

Paso 2: Inyectar la URL de Collaborator

1. Volver a la pestaña **Repeater**.
2. Sustituir la URL por la dirección única de **Collaborator** que acabamos de generar.

Solicitud Maliciosa:

```
POST /api/v1/download_avatar con payload {"url": "http://xyz.burpcollaborator.net/test"}
```

3. Hacer clic en **"Send"**.

Paso 3: Monitorear el Collaborator

1. Regresar a la pestaña **Burp Collaborator Client**.
2. Analizar los resultados:

Vulnerable: Si **Collaborator recibe una o más interacciones DNS o HTTP** del servidor de la API. Esto confirma que el servidor realizó la petición al dominio controlado por el atacante.

Seguro: Collaborator no recibe interacciones. Esto sugiere que el servidor solo permite la conexión a dominios en una *whitelist* o que no ejecutó la petición.

4.3.2.8. API8 Security Misconfiguration

Objetivo

Identificar y documentar configuraciones inseguras a nivel de infraestructura, servidor web, *framework* o capa de aplicación que exponen información sensible o relajan los controles de seguridad predeterminados.

Prerrequisitos

1. **Endpoints de Prueba:** Varios *endpoints* para verificar la consistencia de la configuración (un GET simple, un POST con validación, un *endpoint* inexistente).

Paso a Paso para la Explotación con Burp Suite (Proxy & Repeater)

El ataque se divide en tres escenarios principales: Manejo de Errores, Exposición de Headers y Configuración de CORS/Métodos.

Escenario 1: Exposición de Información por Manejo de Errores

El objetivo es forzar al servidor a fallar para que revele información interna (rutas de archivo, versiones de *frameworks*, *stack traces*).

Paso 1: Forzar una Validación o Error

1. Ejecutar una petición normal (POST o PUT) que requiera validación (ej. crear un usuario con un campo requerido faltante o un tipo de dato incorrecto).

2. Capturar la solicitud en el **Proxy** de Burp y envíarla a **Repeater**.

3. En Repeater, modificar el *payload* para forzar un error obvio (ej. inyectar un *string* muy largo en un campo numérico, o inyectar caracteres especiales, como una comilla simple ').

4. Hacer clic en **"Send"** y analizar la respuesta.

Paso 2: Forzar un Error de Ruta (Endpoint Inexistente)

1. Tomar cualquier solicitud válida en Burp y envíarla a **Repeater**.

2. Modificar la URL a una ruta que se sabe que no existe (ej. `/api/v1/non_existent_path`).

3. Hacer clic en **"Send"** y analizar la respuesta.

Vulnerable: Código **500 Internal Server Error** o **400 Bad Request**, y el cuerpo contiene rutas de archivo completas, nombres de bases de datos, consultas SQL (SELECT * FROM...) o un stack trace (pila de llamadas). La API está mal configurada para exponer detalles de la implementación, facilitando futuros ataques.

Seguro: Códigos de error apropiados (**400**, **404**, **405**) y el cuerpo solo contiene un mensaje genérico (ej. "Error inesperado" o "La ruta no existe"). Los mensajes de error no revelan información interna.

Escenario 2: Análisis de Headers de Respuesta y Versiones

El objetivo es encontrar *headers* faltantes o *headers* que revelan versiones de software.

Paso 1: Recolectar Headers

1. Realizar una petición GET simple a cualquier *endpoint* (ej. /api/v1/status).
2. Analizar la respuesta en el **Proxy** o **Repeater** de Burp.

Paso 2: Revisar la Exposición de Versiones

1. Busca *headers* que expongan versiones específicas:
    a. Server: (ej. Apache/2.4.41 o nginx/1.18.0)
    b. X-Powered-By: (ej. ASP.NET, Express)
    c. Nombres de *frameworks* o lenguajes (ej. X-AspNet-Version).

    Vulnerable: Si alguno de estos *headers* está presente, revela el software subyacente.

Paso 3: Revisar Headers de Seguridad Críticos

1. Buscar la **ausencia** o **mala configuración** de los siguientes *headers* de seguridad:
    a. Strict-Transport-Security (HSTS): Faltante, permitiendo ataques de *downgrade*.
    b. Content-Security-Policy (CSP): Faltante o configurado incorrectamente.

c. X-Frame-Options: Faltante (permite *clickjacking*).

Vulnerable: Si alguno de estos *headers* está ausente o mal configurado.

Escenario 3: Configuración Insegura de CORS y Métodos HTTP

Paso 1: Prueba de CORS (Cross-Origin Resource Sharing)

1. Tomar una solicitud GET en **Repeater**.

2. Añadir el *header*: Origin: https://www.atacante.com (un dominio malicioso).

3. Si la API devuelve un *header* de respuesta Access-Control-Allow-Origin: * o Access-Control-Allow-Origin: https://www.atacante.com.

Vulnerable: Esto permite que el dominio del atacante (https://www.atacante.com) acceda al contenido de la API con las credenciales del usuario, explotando la **Configuración Insegura de CORS**.

Paso 2: Prueba de Métodos HTTP No Autorizados

1. Tomar una solicitud **GET** simple en **Repeater**.

2. Cambiar el método a **OPTIONS** y envíar. La respuesta debería listar los métodos permitidos (ej. Allow: GET, POST).

3. Cambiar el método a **TRACE** y envía.

Vulnerable: Si el servidor responde con **200 OK** a una petición TRACE, la configuración permite el método TRACE, lo que puede utilizarse para robar *cookies* de sesión (*Cross-Site Tracing*).

4. Cambiar el método a **PUT** o **DELETE** en un *endpoint* que solo debería soportar GET o POST.

Vulnerable: Si el servidor responde con **200 OK** o **405 Method Not Allowed** pero revela información, o si un método de bajo consumo (ej. GET) funciona con un método de alto riesgo (DELETE), confirmando que las restricciones de método no se aplican correctamente.

4.3.2.9. API9 Gestión de Inventario

Objetivo

Identificar y explotar endpoints obsoletos, versiones API sin mantenimiento, o funciones internas no documentadas (ej. /v1/users, /admin/status). Estos activos no gestionados suelen carecer de los controles de seguridad aplicados a las versiones actuales.

Prerrequisitos

1. **Documentación de la API:** Conocer la versión actual y documentada (ej. v3).

2. **Credencial de Prueba:** Un **Usuario A** con bajos o nulos privilegios (para verificar si los *endpoints* obsoletos ignoran la autorización).

Paso a Paso para la Explotación con Burp Suite (Intruder)

La prueba se centra en dos escenarios de descubrimiento para mitigar el riesgo **API9: Improper Inventory Management**.

Escenario 1: Enumeración de Versiones Obsoletas

El objetivo es determinar si las versiones anteriores de la API todavía están operativas y si su seguridad es inferior.

Paso 1: Interceptar y Preparar la Solicitud

1. En el navegador, realizar una solicitud válida a la **versión actual y documentada** (ej. GET /api/v3/users/100).

2. Capturar la solicitud en el **Proxy** de Burp y envíarla a **Intruder**.

Paso 2: Configurar la Posición del Ataque

1. Ir a la pestaña **Intruder**.

2. En **Positions**: Marcar únicamente el número de la versión dentro de la ruta (/v3/) como posición de ataque.

3. **Attack Type:** Seleccionar **Sniper**.

Paso 3: Cargar el Payload de Versiones

1. En **Payloads**:

    a. **Payload Type:** Seleccionar **Numbers** o **Simple List**.

    b. **Payload Options (Números):** Generar una secuencia para probar versiones anteriores y futuras (ej. de 1 a 5, con un paso de 1).

    c. **Payload Options (Lista):** Incluir formatos alternativos (ej. v1, v2, v3, v4, v2_legacy, v1_old).

Paso 4: Ejecutar y Analizar el Inventario

1. Hacer clic en **"Start attack"**.

2. Analizar la tabla de resultados buscando códigos de estado **200 OK** para versiones no documentadas (ej. /v1/).

3. **Prueba de Seguridad (BOLA/BFLA en Viejas Versiones):** Si /v1/users/100 devuelve **200 OK**, repetir la prueba **API1 (BOLA)** y **API5 (BFLA)** en esa versión antigua, usando el token del Usuario A (bajo privilegio).

Vulnerable: Si la versión obsoleta no implementa correctamente los filtros de autorización (API1/API5).

Escenario 2: Fuzzing de Endpoints Ocultos

El objetivo es encontrar *endpoints* de prueba, *backups* o de administración que no se han retirado.

Paso 1: Configurar la Petición de Fuzzing

1. Tomar una solicitud GET simple en el **Proxy** y envíarla a **Intruder**.

Paso 2: Definir la Posición y Cargar el Diccionario

1. En **Positions**: Marca la **última parte de la ruta** como posición de ataque.
2. En **Payloads**:
    a. **Payload Type:** Seleccionar **Simple List**.
    b. **Cargar Lista:** Cargar un *wordlist* específico para descubrimiento de API que incluya términos comunes (ej. admin, dev, test, backup, status, health, monitor, logs, config).

Paso 3: Ejecutar y Analizar Fugas

1. Hacer clic en **"Start attack"**.

2. Analiza**r** la tabla de resultados buscando discrepancias:

    a. **Códigos de estado diferentes:** (ej. **200 OK** o **3xx Redirección**), en lugar del esperado **404 Not Found**.

    b. **Longitud diferente:** Un cambio significativo en la longitud del cuerpo de la respuesta para un *endpoint* en particular.

3. **Prueba de Impacto:** Si se encuentra un *endpoint* oculto (ej. /api/v3/logs), intenta acceder a él con el token del Usuario A.

    Vulnerable: Si el *endpoint* expone archivos de registro, datos internos o permite la ejecución de acciones sin requerir privilegios de administrador, se confirma una falla de **API9** que conduce a fallas de **API5** o **API8**.

4.3.2.10. API10 Consumo Inseguro de APIs

No se aplica esta prueba debido a que la API no consume datos o servicios de APIs de terceros

4.3.3. Fase de Reconocimiento

A continuación, se describen los resultados de esta fase:

- **Recopilación:** Se utilizó Swagger/OpenApi en la ruta /api-docs. En donde consta una descripción completa de la API, sus endpoints.

Tabla 7:

Mapa de la API

Endpoint / Método	**Parámetros (Path/Query/Body)**	**Roles Permitidos (API5)**	**Observaciones de Seguridad (API3/API9)**	**Resultado del Descubrimiento Dinámico**
GET /api/v3/user/{id}	**Path:** user_id (numérico) **Query:** expand (opcional)	**Standard**, Premium, Admin	**Salida:** Expone internal_client_id, password_hash. **Riesgo:** Exposición excesiva de datos (**API3**).	GET /v1/user/10 está activo y devuelve datos. (**API9** - Versión obsoleta).
POST /api/v3/order	**Body:** product_id, quantity, shipping_address, price (cliente-controlado)	**Standard**, Premium	**Entrada:** Acepta price en el body. **Riesgo:** Posible *Mass Assignment* o manipulación de precio.	Se observó que POST /v2/order/create está activo, pero no documentado.
PUT /api/v3/profile/edit	**Body:** name, email, preferences, role (no esperado)	**Standard**, Premium	**Entrada:** Se acepta un campo role en la carga útil que es ignorado por la documentación. **Riesgo:** Posible *Mass Assignment* (**API3**).	El tráfico en vivo reveló el parámetro de *query* ?lang=es.
GET /api/v3/orders	**Query:** limit, offset	**Standard**, Premium, Admin	**Control:** Los parámetros limit y offset no tienen un máximo definido en la especificación. **Riesgo:** Consumo de	El endpoint /admin/status está activo y no requiere autenticación. (**API9**).

			recursos (**API4** - Pendiente de prueba).	
DELETE /api/v3/user/{id}	**Path:** user_id (numérico)	**Admin**	**Acceso:** Solo rol Admin, crítico. **Riesgo:** BFLA si es accesible por otro rol (**API5**).	El user_id en el tráfico en vivo es secuencial. **Riesgo:** IDs predecibles (**API1**).

4.3.4. FASE III Reporte

Para el reporte de los resultados, tomaremos como referencia NIST y OWASP, haciendo énfasis en las vulnerabilidades identificadas, su severidad, el riesgo relacionado y las recomendaciones para su cierre. A continuación, se describe con mayor detalle lo que contemplará el reporte de resultados:

I. Resumen Ejecutivo

- **Alcance:** Breve descripción de la API o el entorno evaluado.
- **Período de Prueba:** [Fecha de inicio] - [Fecha de fin].
- **Herramientas Utilizadas:** Burp Suite Professional, OWASP ZAP.
- **Puntuación de Riesgo General:** Nivel de riesgo consolidado (**Alto**).
- **Conclusiones Clave:** Declaración concisa de la vulnerabilidad más crítica (ej. Fallo crítico en el control de acceso a nivel de objeto).

II. Resumen por Nivel de Riesgo (Risk Rating Summary)

Tabla 8:

Cantidad de vulnerabilidades encontradas por su nivel de riesgo

Nivel de Riesgo	Número de Vulnerabilidades	% del Total
Crítico	[Número]	[%]
Alto	[Número]	[%]
Medio	[Número]	[%]
Bajo	[Número]	[%]
Informativo	[Número]	[%]

III. Detalle de Vulnerabilidades

En esta sección se describa con mayor detalle cada hallazgo encontrado.

Hallazgo # X: Nombre de la Vulnerabilidad

Tabla 9:

Nombre de la Vulnerabilidad

Sección	Contenido Requerido
Categoría OWASP API	**[API1:2023 - Broken Object Level Authorization (BOLA)]**

Nivel de Riesgo	**[Alto]**
CVSS v3.1	[Puntuación y Vector CVSS, 8.3 / AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N]
Endpoint Afectado	[Método HTTP] https://developer.ebay.com/api-docs/sell/fulfillment/resources/order/methods/getOrders
Parámetros Afectados	[orderId en la ruta, *token* de sesión]
Herramienta de Detección	[Burp Suite Repeater] o [OWASP ZAP Active Scan - Regla X]

Descripción y Evidencia Técnica

- **Descripción de la Vulnerabilidad:** Explicación clara de por qué ocurre el fallo (El servidor no valida que el orderId pertenezca al usuario autenticado).

- **Evidencia:** Incluye la solicitud y respuesta HTTP completa que demuestra la vulnerabilidad.

Recomendaciones de Corrección

- Instrucciones para que el equipo de desarrollo solucione el problema.

CAPITULO 5

5. ANÁLISIS Y RESULTADOS

5.1. Resumen Ejecutivo

5.2. Alcance: Descripción de la API Analizada

La API evaluada se encuentra implementada bajo una arquitectura de microservicios expuesta mediante un gateway, lo que permite escalabilidad y desacoplamiento de funciones críticas. El entorno de despliegue se basa en infraestructura virtualizada, soportado sobre contenedores Dockers, con integración a una base de datos relacional PostgreSQL. Este contexto tecnológico implica consideraciones específicas de seguridad, tales como la protección de datos en el entorno virtualizado, la segmentación de redes virtuales, la configuración de balanceadores de carga (en producción) y la protección frente a ataques de denegación distribuida de servicio (DDoS).

La evaluación se limita a los endpoints de la API RESTful virtualizados en el entorno de pruebas, sin incluir análisis del front-end ni de componentes externos a la infraestructura documentada.

Período de Prueba: 25 de septiembre de 2025 – 4 de diciembre de 2025

Herramientas Utilizadas: Burp Suite Professional, OWASP ZAP.

Puntuación de Riesgo General: Alto.

5.3. Resumen por Nivel de Riesgo

5.4. Hallazgos por Vulnerabilidad OWASP

A continuación, se realiza una presentación detallada de cada vulnerabilidad identificada, incluyendo: descripción de la Vulnerabilidad, evidencia de detección (capturas de pantalla, logs de tráfico, etc.), impacto en la Seguridad, riesgo Asociado (bajo, medio, alto).

Resumen por Nivel de Riesgo

Tabla 100:

Cantidad de vulnerabilidades encontradas - OWASP ZAP por su nivel de riesgo

Nivel de Riesgo	Número de Vulnerabilidades	% del Total
Alto	0	0%
Medio	4	31%
Bajo	6	46%
Informativo	3	23%

5.4.1. Hallazgo # 1: CSP: Failure to Define Directive with No Fallback

Tabla 11:

Failure to Define Directive with No Fallback

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)

Nivel de Riesgo	**Medio**
CVSS v3.1	No tiene un único vector o puntuación CVSS v3.1 predefinida, ya que no es una vulnerabilidad de código directa, sino un fallo de configuración que reduce la mitigación contra otros ataques
Endpoint Afectado	POST https://localhost:3000/registro_empleado
Parámetros Afectados	Ninguno específicamente.
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 10038: Falta de Content-Security-Policy)

Descripción de la Vulnerabilidad: La Content Security Policy falla al definir una de las directivas que no tiene fallback. Omitirlas/excluirlas es lo mismo que permitir todo.

Recomendaciones de Corrección

1. **Definir la Directiva de Respaldo (default-src)**

   El fallo ocurre porque al faltar una directiva específica (font-src), el navegador necesita una regla general. La directiva **default-src** actúa como el respaldo universal (*fallback*).

   Por tanto, es necesario configurar default-src con el valor de **self** para asegurar que solo permita cargar recursos que provengan del mismo origen (dominio, protocolo y puerto) de la página.

2. **Fortalecer y Limitar las Fuentes de Script**

   El mayor riesgo de este fallo es el XSS. Debes asegurarte de que script-src no use fuentes inseguras, para lo cual se debe eliminar 'unsafe-inline' y 'unsafe-eval al configurarlo y usar hashes ('sha256-...') o nonces ('nonce-...'), ya que es la forma más segura de permitir scripts en línea o específicos, en este caso el navegador solo ejecutará scripts que coincidan con el hash o nonce.

5.4.2. Hallazgo # 2: Cabecera Content Security Policy (CSP) no configurada

Tabla 12:

Cabecera Content Security Policy (CSP) no configurada

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Medio**
CVSS v3.1	No tiene un único vector o puntuación CVSS v3.1 predefinida, ya queno es una vulnerabilidad de código que se explota directamente, sino un **fallo de control de seguridad** que habilita la explotación de otras vulnerabilidades (principalmente **Cross-Site Scripting - XSS**).
Endpoint Afectado	GET https://192.168.56.20:3000/api-docs POST https://192.168.56.20:3000/registro_persona POST https://192.168.56.20:3000/registro_plan_membresia

Parámetros Afectados	Ninguno específicamente.
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 10038: Falta de Content-Security-Policy)

Descripción de la Vulnerabilidad

La Política de seguridad de contenido (CSP) es una capa adicional de seguridad que ayuda a detectar y mitigar ciertos tipos de ataques, incluidos Cross Site Scripting (XSS) y ataques de inyección de datos. Estos ataques se utilizan para todo, desde el robo de datos hasta la desfiguración del sitio o la distribución de programa maligno (malware). CSP proporciona un conjunto de encabezados HTTP estándar que permiten a los propietarios de sitios web declarar fuentes de contenido aprobadas que los navegadores deberían poder cargar en esa página; los tipos cubiertos son JavaScript, CSS, marcos HTML, fuentes, imágenes y objetos incrustados como applets de Java, ActiveX, archivos de audio y video.

Recomendaciones de Corrección

La vulnerabilidad de la Cabecera Content Security Policy (CSP) no configurada se remedia implementando y ajustando correctamente el header Content-Security-Policy para que sea estricto y completo.

El proceso de remediación debe ser metódico, ya que una política demasiado estricta puede romper la funcionalidad del sitio, mientras que una política demasiado laxa no ofrece protección.

1. Implementación Inicial y Mínima (Modo Reporte)

Desplegar una política que registre las violaciones sin bloquear el contenido, lo que permitirá auditar qué recursos se están cargando.

1. **Header de Reporte:** Implementar la política usando Content-Security-Policy-Report-Only. Esto permite que el navegador **solo envíe reportes** de violaciones a un *endpoint* definido (report-uri o report-to), sin bloquear el contenido.

HTTP Content-Security-Policy-Report-Only: default-src 'self'; report-uri /csp-reporting-endpoint;

2. **Monitoreo:** Analizar el *endpoint* de reporte para identificar todas las fuentes de contenido (scripts, estilos, imágenes, etc.) que la aplicación legítimamente necesita, especialmente las fuentes de terceros (CDNs).

2. Definición de la Política Estricta (default-src y Orígenes Confiables)

Una vez que se sabe qué recursos se necesitan, definir la política en el *header* final Content-Security-Policy.

1. **Definir el Respaldo Estricto:** Define **default-src 'self'** como la base. Esto garantiza que cualquier recurso no listado explícitamente solo se pueda cargar desde el propio dominio.
2. **Definir Fuentes Específicas:** Sobrescribir default-src para los tipos de contenido que deben cargarse desde otros dominios.
    a. **scripts:** Usar **script-src** y enumerar solo los dominios confiables.
    b. **estilos:** Usar **style-src** para hojas de estilo externas.
    c. **Imágenes:** Usar **img-src** para logotipos o imágenes de servicios externos.

3. Refuerzo contra XSS

Dado que la ausencia de CSP permite XSS, la protección más fuerte debe centrarse en **script-src**, para lo cual se debe eliminar 'unsafe-inline' y 'unsafe-eval al configurarlo y usar hashes ('sha256-...') o nonces ('nonce-...'), ya que es la forma más segura de permitir scripts en línea o específicos, en este caso el navegador solo ejecutará scripts que coincidan con el hash o nonce.

4. Bloqueo de Funciones Peligrosas

Añadir directivas para bloquear explícitamente funciones que pueden ser explotadas:

- **object-src 'none':** Bloquea la carga de *plugins* antiguos como Flash.
- **base-uri 'self':** Evita que atacantes cambien la ubicación de scripts.
- **frame-ancestors 'none':** Evita que el contenido sea incrustado en *iframes* de otros sitios (mitigación de *clickjacking*).

5.4.3. Hallazgo # 3: Divulgación de error de aplicación

Tabla 13:

Divulgación de error de aplicación

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Medio**

CVSS v3.1	No tiene un único vector o puntuación CVSS v3.1 predefinida, ya que su gravedad depende directamente de la **cantidad y criticidad de la información interna expuesta**. No es el error en sí mismo, sino lo que revela, lo que determina el riesgo.
Endpoint Afectado	POST https://localhost:3000/registro_empleado POST https://192.168.56.20:3000/registro_empleado
Parámetros Afectados	Ninguno específicamente.
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 10038: Falta de Content-Security-Policy)

Descripción de la Vulnerabilidad

Esta página contiene un mensaje de error/advertencia que podría revelar información sensible como la ubicación del archivo que produjo la excepción no controlada. Esta información puede ser usada para lanzas futuros ataques contra la aplicación web. La alerta podría ser una falso positivo si el mensaje de error es encontrado dentro de una página de documentación.

Recomendaciones de Corrección

Para remediar la vulnerabilidad de **Divulgación de Error de Aplicación** (Application Error Disclosure) y proteger la API de ataques de reconocimiento avanzado se debe **capturar los errores internos** y devolver respuestas **genéricas, controladas y estandarizadas** al

cliente, mientras se asegura que los detalles críticos se registren de forma segura en el *backend*.

1. Desactivar Respuestas Verbosas de la Infraestructura

El primer paso es evitar que el framework o servidor web revele información por defecto.

Servidor Web/Proxy: Configurar servidores web como Nginx o Apache (o el gateway de la API) para que no muestren páginas de error por defecto ("Internal Server Error" con la versión del servidor). Redirigir todos los errores 5xx a una página estática simple.

Configuración del Framework: En entornos de producción, asegurarse de que el modo de depuración (*debug mode*) esté desactivado. El *debug mode* suele ser la causa principal de la exposición de trazas de pila.

2. Implementar Manejo de Errores Centralizado (Error Handling)

Todos los errores deben ser interceptados por una capa de manejo de errores dedicada antes de que salgan de la API.

Crear un *Middleware* Global: Implementar un componente (*middleware* o *exception handler*) que capture todas las excepciones no manejadas o errores del sistema.

Captura de Excepciones: Este componente debe ser capaz de distinguir entre:

- **Errores del Cliente (4xx):** Deben devolver mensajes útiles (400 Bad Request con el campo JSON específico que falló).
- **Errores del Servidor (5xx):** Nunca deben devolver la traza de pila.

3. Estandarizar las Respuestas de Error (Sanitización)

La API debe devolver solo la información esencial y estandarizada al cliente.

Tabla 14:

Tipo de error y su manejo

Tipo de Error	Código HTTP	JSON de Respuesta Estándar	Explicación
Fallo Crítico (Backend)	500 Internal Server Error	{"error_code": "API-5001", "message": "Ocurrió un error inesperado."}	**Nunca incluir la traza de pila.** El código API-5001 permite al equipo de *backend* encontrar la traza en los logs.
Recurso no encontrado	404 Not Found	{"error_code": "RES-4040", "message": "El recurso solicitado no existe."}	Evitar mensajes como "No se encontró el ID de usuario 'X' en la tabla de la base de datos 'users'."
Autorización fallida	403 Forbidden	{"message": "Acceso denegado."}	El mensaje debe ser genérico para evitar dar pistas sobre el estado de la lógica interna.

4. Logging Detallado en el Backend (Logging Seguro)

La información crítica expuesta en la traza de pila es vital para el equipo de desarrollo, pero debe ser almacenada de forma segura.

Registrar la Traza: La función de manejo de errores debe registrar la traza de pila completa (incluyendo variables, rutas de archivos y versiones) en un sistema de *logging* centralizado (ELK Stack, Splunk, Cloudwatch).

Aislamiento: Asegurarse de que los logs solo sean accesibles por el personal de operaciones y desarrollo a través de canales seguros y autenticados, y que nunca se escriban en la respuesta HTTP.

5.4.4. Hallazgo # 4: Falta de cabecera Anti-Clickjacking

Tabla 15:

Falta de cabecera Anti-Clickjacking

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Medio**
CVSS v3.1	No tiene una puntuación CVSS v3.1 fija, ya que, por sí misma, no es la vulnerabilidad de ejecución, sino el fallo de control que facilita un ataque de Clickjacking o UI Redress.
Endpoint Afectado	GET https://192.168.56.20:3000/api-docs
Parámetros Afectados	Ninguno específicamente.
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 10038: Falta de Content-Security-Policy)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

La respuesta no protege contra ataques de "ClickJacking". Debes incluir Content-Security-Policy con la directiva "frame-ancestors" o X-Frame-Options.

- **Evidencia:** NA

Recomendaciones de Corrección

Para remediar la vulnerabilidad de la Falta de cabecera Anti-Clickjacking se debe configurar un *header* HTTP en el servidor web, el *gateway* de la API, o el código de la aplicación (manejo de las respuestas HTTP).

El objetivo es asegurar que la página no pueda ser incrustada en *iframes* de sitios maliciosos.

Usar Content-Security-Policy: frame-ancestors

Esta es la forma moderna de mitigar el *clickjacking* y se incluye dentro de la política de Content Security Policy (CSP).

Bloqueo Total: Configurar la directiva para prohibir totalmente la incrustación.

Content-Security-Policy: frame-ancestors 'none'

Fuentes Específicas: Especificar los dominios exactos que **pueden** incrustar la página.

Content-Security-Policy: frame-ancestors 'self' https://trusted.domain.com

5.4.5. Hallazgo # 5: Divulgación de Marcas de Tiempo - Unix

Tabla 16:

Divulgación de Marcas de Tiempo - Unix

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Bajo**
CVSS v3.1	No tiene una puntuación CVSS v3.1 fija, ya que el riesgo reside en el **potencial de reconocimiento** que le otorga al atacante, no en el impacto directo en la integridad o disponibilidad del sistema.
Endpoint Afectado	[GET] https://192.168.56.20:3000/api-docs
Parámetros Afectados	Ninguno específicamente.
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 10038: Falta de Content-Security-Policy)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

En este caso, el servidor no envía la cabecera HTTP Content-Security-Policy, la cual es esencial para limitar las fuentes de contenido que puede cargar el navegador

Recomendaciones de Corrección

Para corregir este problema, se debe implementar una Content-Security-Policy robusta

y restrictiva en todas las respuestas HTTP.

1. Añadir una cabecera CSP mínima recomendada
2. Asegurar la implementación en el servidor (Express.js)
3. Reforzar con otros encabezados faltantes (opcional pero recomendado)
    a. X-Frame-Options: DENY
    b. X-Content-Type-Options: nosniff
4. Strict-Transport-Security
5. Revisar uso de Swagger (api-docs), Swagger UI suele cargar scripts externos, por lo que puede requerir una CSP más específica

5.4.6. Hallazgo # 6: Divulgación de error de aplicación

Tabla 17:

Divulgación de error de aplicación

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Bajo**
CVSS v3.1	No tiene una puntuación **CVSS v3.1** única y fija, ya que su gravedad depende directamente de la **cantidad y criticidad de la información interna expuesta**. El riesgo se evalúa basándose en el **potencial de reconocimiento** que la divulgación le otorga al atacante.
Endpoint Afectado	[POST] https://192.168.56.20:3000/registro_empleado

	[POST] https://localhost:3000/registro_empleado
Parámetros Afectados	No parámetros específicos. El error ocurre debido al procesamiento incorrecto del cuerpo de la petición (POST).
Herramienta de Detección	OWASP ZAP – Active Scan (Plugin Id 90022 – Application Error Disclosure)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

El servidor expone mensajes de error internos al cliente. ZAP detectó que las respuestas incluyen información sensible, como:

- Mensajes de excepción no manejada
- Stack traces parciales
- Errores explícitos del motor de ejecución de JavaScript (Node.js)
- Indicadores del punto exacto donde falló la lógica

Recomendaciones de Corrección

1. Implementar manejo de errores centralizado (Express.js)
2. Nunca enviar errores crudos al cliente
3. Evitar respuestas como:
    a. SyntaxError: Unexpected token
    b. TypeError: Cannot read property 'x' of undefined
    c. Stack traces
    d. Mensajes del motor de Node.js
4. Validar entradas antes de procesar, Usar Joi, Zod o middleware de validación para evitar

errores de parsing.

5. Registrar errores en backend, con herramientas como: Winston, Morgan, ELK stack
6. Revisar puntos del código donde el parser JSON puede fallar Los mensajes "Unexpected token" generalmente provienen de:
    a. JSON.parse() incorrecto
    b. body-parser fallando por tipos inválidos
    c. Middleware que no valida contenido entrante

5.4.7. Hallazgo # 7: El servidor divulga información mediante el encabezado HTTP X-Powered-By

Tabla 18:

El servidor divulga información mediante el encabezado HTTP X-Powered-By

Sección	Contenido Requerido
Categoría OWASP API	API9:2023 – Improper Assets Management (Gestión Incorrecta de Activos)
Nivel de Riesgo	**Bajo**
CVSS v3.1	
Endpoint Afectado	En total 68 instancias, que incluyeron X -Powered-By: Express: [DELETE] https://192.168.56.20:3000/cancelar_reserva/10 [DELETE] https://192.168.56.20:3000/eliminar_empleado/id [GET] https://192.168.56.20:3000/membresias [GET] https://192.168.56.20:3000/maquinas

	[GET] https://localhost:3000/api-docs.json [POST] https://localhost:3000/registro_empleado [PUT] https://localhost:3000/modificar_empleado/id
Parámetros Afectados	Ninguno ya que la vulnerabilidad está en los encabezados de respuesta, no depende de parámetros.
Herramienta de Detección	OWASP ZAP – Passive/Active Scan (Plugin Id 10037 – X-Powered-By Header Disclosure)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

El servidor expone en la cabecera HTTP:

Esto revela el framework backend utilizado: Express.js (Node.js).

La exposición de información sobre el framework (como la versión de Express) aumenta el riesgo de ataques dirigidos, ya que permite identificar vulnerabilidades conocidas, facilita la explotación de fallos como *path traversal* o *template injection*, y ayuda a deducir configuraciones internas del servidor.

Recomendaciones de Corrección

1. Deshabilitar el encabezado X-Powered-By en Express
2. Usar Helmet (si no está instalado), ya que Helmet automáticamente elimina el encabezado.
3. Revisar módulos que vuelvan a agregarlo, Algunos frameworks, plugins o middlewares

pueden volver a inyectarlo.

5.4.8. Hallazgo # 8: Falta encabezado X-Content-Type-Options

Tabla 19:

Falta encabezado X-Content-Type-Options

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Bajo**
CVSS v3.1	
Endpoint Afectado	En total 17 instancias • [GET] https://192.168.56.20:3000/api-docs • [GET] https://192.168.56.20:3000/maquinas • [GET] https://192.168.56.20:3000/membresias • [GET] https://192.168.56.20:3000/membresias_por_plan/10 • [GET] https://192.168.56.20:3000/pagos_socio/id • [GET] https://localhost:3000/api-docs.json
Parámetros Afectados	Ninguno (la falta es en encabezados de respuesta; afecta a todas las respuestas del endpoint).
Herramienta de Detección	OWASP ZAP – Passive/Active Scan (Plugin Id 10021 – X-Content-Type-Options missing)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

La cabecera X-Content-Type-Options: nosniff no está presente en las respuestas HTTP. Sin esta cabecera, navegadores (especialmente versiones antiguas o mal configuradas) pueden realizar MIME-sniffing — interpretar un recurso con un tipo distinto al declarado por Content-Type. Esto puede permitir la ejecución de JavaScript o la interpretación de contenido como HTML cuando en realidad es un archivo descargable, facilitando vectores de ataque tipo XSS o exfiltración en contextos específicos.

- **Evidencia:** Incluye la solicitud y respuesta HTTP completa que demuestra la vulnerabilidad.

Recomendaciones de Corrección

1. Añadir la cabecera X-Content-Type-Options: nosniff en todas las respuestas, incluyendo páginas de error (401, 403, 500) y recursos estáticos.
2. Verificar endpoints estáticos y rutas de documentación (Swagger / api-docs) — algunos UIs cargan recursos que podrían necesitar Content-Type correcto; asegurar headers en esas respuestas.
3. Pruebas post-implementación: volver a ejecutar ZAP o Burp y comprobar que la cabecera aparece en todas las respuestas (incluyendo errores y contenido estático). ZAP debería dejar de reportar el plugin Id 10021.
4. Política de seguridad: documentar la obligación de incluir encabezados de seguridad en la guía de despliegue/operaciones y añadir comprobaciones a pipelines (SAST/DAST) o checks HTTP en CI.

5.4.9. Hallazgo # 9: Revelación de IP privada

Tabla 20:

Revelación de IP privada

Sección	Contenido Requerido
Categoría OWASP API	API9:2023 – Improper Assets Management (Gestión Incorrecta de Activos)
Nivel de Riesgo CVSS v3.1	**Bajo**
Endpoint Afectado	[GET] https://localhost:3000/api-docs.json
Parámetros Afectados	Ninguno.
Herramienta de Detección	La IP privada aparece dentro del **cuerpo JSON** de la documentación de la API.
	OWASP ZAP – Passive Scan (Plugin Id 2 – Private IP Disclosure)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

La API expone información sensible de la infraestructura al incluir una **dirección IP privada** dentro del cuerpo de la respuesta HTTP

Este tipo de información puede ayudar a un atacante a:

- Identificar la red interna.

- Mapear activos internos relacionados.
- Realizar escaneo dirigido si obtiene acceso a la red privada.
- Comprender la arquitectura del entorno (servidor, redes virtuales, puertos).

Recomendaciones de Corrección

1. Eliminar IP privadas de la documentación de la API (Swagger/OpenAPI) Edita el archivo openapi.json o swagger.json para reemplazar la IP interna por una variable de entorno o URL pública.

2. Evitar hardcodear direcciones internas en el backend, se debe utilizar variables de entorno

3. Revisar mensajes de error y respuestas generadas dinámicamente, asegúrate de que ningún error incluya rutas internas como 10.x.x.x o 172.16.x.x

4. Filtrar salida en producción, configurar Swagger para que en producción NO muestre servidores internos.

5.4.10. Hallazgo # 10: Strict-Transport-Security Header No Establecido

Tabla 21:

Strict-Transport-Security Header No Establecido

Sección	Contenido Requerido
Categoría OWASP API	API8:2023 – Security Misconfiguration (Mala Configuración de Seguridad)
Nivel de Riesgo	**Bajo**
CVSS v3.1	

Endpoint Afectado	[DELETE] https://192.168.56.20:3000/cancelar_reserva/10 [GET] https://192.168.56.20:3000/api-docs [GET] https://192.168.56.20:3000/membresias [GET] https://192.168.56.20:3000/maquinas [POST] https://192.168.56.20:3000/registro_empleado [PUT] https://localhost:3000/modificar_empleado/id [GET] https://localhost:3000/api-docs.json n
Parámetros Afectados	Ninguno. El problema corresponde a configuración del servidor en la respuesta HTTP, no a parámetros.
Herramienta de Detección	OWASP ZAP – Passive/Active Scan (Plugin Id 10035 – Strict-Transport-Security Header Missing)

Descripción y Evidencia Técnica

Descripción de la Vulnerabilidad

El encabezado HSTS obliga a los navegadores a conectarse siempre por HTTPS incluso si el usuario intenta acceder vía HTTP.

Sin este encabezado, un atacante puede explotar:

- SSL stripping

- Man-in-the-Middle (MITM) en redes públicas
- Redirecciones inseguras antes de cargar HTTPS

Se detectó que ninguna respuesta de la API incluye el encabezado Strict-Transport-Security, lo que es una mala práctica de configuración del servidor.

Recomendaciones de Corrección

1. Agregar el encabezado HSTS correctamente: Strict-Transport-Security

- max-age=31536000 → 1 año obligatorio
- includeSubDomains → protege todos los subdominios
- preload → opción para incluir el dominio en la lista HSTS de navegadores

2. Implementación en Express (Node.js) o O con Helmet (recomendado):

3. Configuración en Nginx y Configuración en Apache

4. Asegurar redirección automática HTTP → HTTPS

5.5. Evaluación de Controles de Seguridad

A continuación, se presenta el análisis de los mecanismos de autenticación, autorización y control de acceso implementados efectuados con Burp Suite

1. Pruebas de Autorización a Nivel de Objeto (BOLA)

Referencia: API1:2023 - Broken Object Level Authorization

1.1. Objetivo de la Prueba

Verificar si la API valida correctamente que el usuario solicitante tenga permisos explícitos para acceder a los recursos de otros usuarios. Dado que la aplicación maneja IDs numéricos secuenciales (ej. id_socio), existe un alto riesgo de enumeración y acceso no autorizado.

1.2. Escenario de Prueba

Se intentará acceder a la información personal y financiera de un "Socio B" utilizando las credenciales válidas de un "Socio A".

1.3. Procedimiento Técnico

1. **Preparación:** Se generan dos usuarios en base de datos: Usuario A (ID: 10) y Usuario B (ID: 11).
2. **Interceptación:** Inicié sesión como Usuario A y navegué a la sección "Mis Pagos". Intercepté la petición HTTP con Burp Proxy.
    a. *Endpoint identificado:* GET /pagos_socio/10
3. **Manipulación:** Envié la solicitud al módulo **Repeater** de Burp Suite. Modifiqué el parámetro de ruta en la URL, cambiando el ID 10 por el ID 11.
4. **Ejecución:** Envié la solicitud manipulada manteniendo la sesión (cookies/headers) del Usuario A.

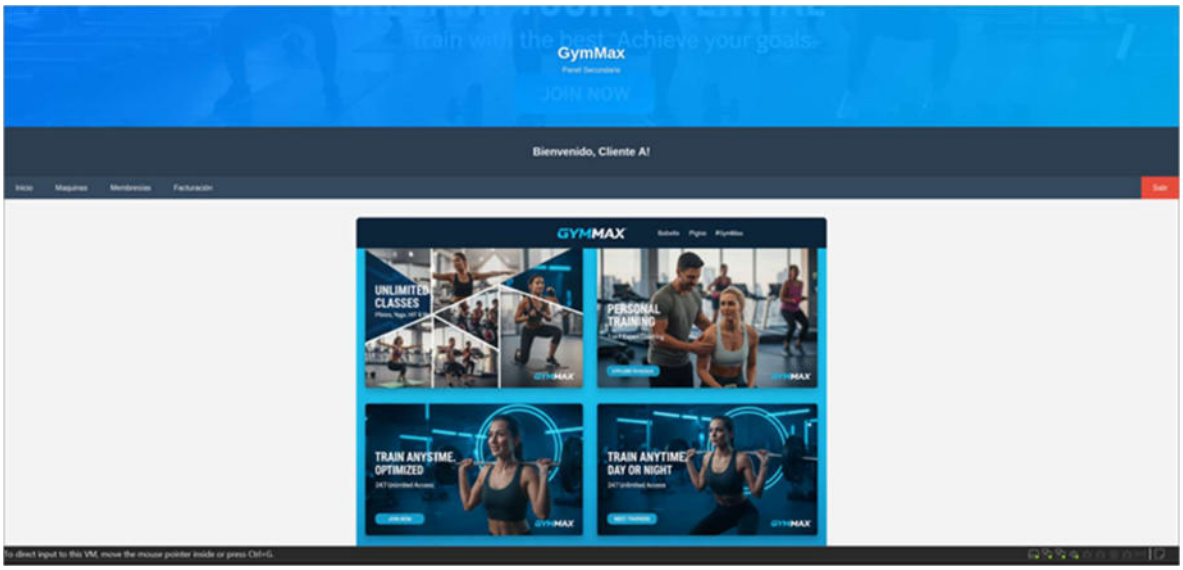
1.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** La API debería rechazar la solicitud con un código de estado 403 Forbidden o 401 Unauthorized, indicando que el Usuario A no es propietario del recurso 11.
- **Resultado Obtenido:**

Loggeado a través de un usuario “A”

Figura 24:

Pantalla Principal Cliente



Vamos a realizar la consulta de pagos de un usuario distinto

Figura 25:

Consulta pagos Cliente

Verificamos que la consulta la trata de realizar a pesar de que esta loggeado a través de un usuario diferente.

Figura 26:

Validación pagos existentes

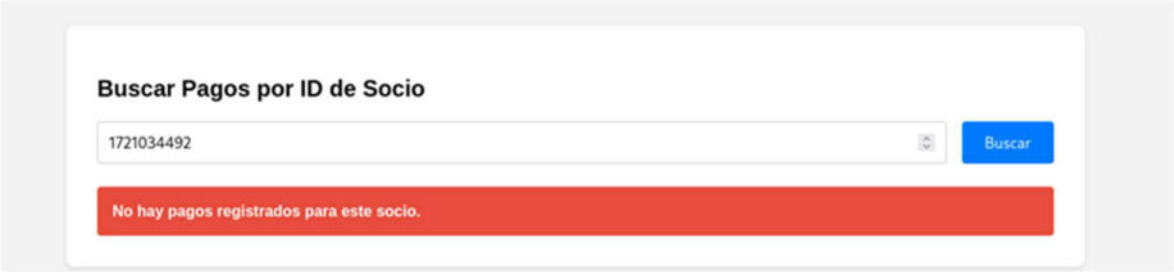
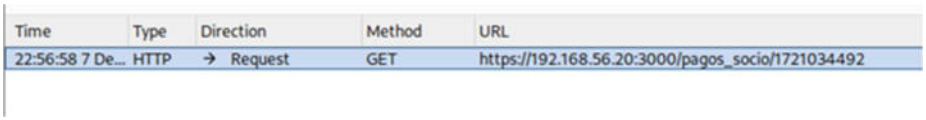


Figura 27:

Captura en Burpsiut del método GET



2. Pruebas de Autenticación y Fuerza Bruta

Referencia: API2:2023 - Broken Authentication

2.1. Objetivo de la Prueba

Evaluar la robustez del mecanismo de login (POST /login_socio) frente a ataques automatizados de adivinación de credenciales, verificando la existencia de limitaciones de tasa (Rate Limiting) o bloqueo de cuentas.

2.2. Escenario de Prueba

Se simulará un ataque de fuerza bruta contra un usuario válido utilizando un diccionario de contraseñas comunes para intentar comprometer la cuenta.

2.3. Procedimiento Técnico

1. **Captura:** Intercepté una petición legítima de inicio de sesión fallido al endpoint /login_socio.
2. **Configuración del Ataque:** Envié la solicitud al módulo **Intruder**.
    a. *Payload Position:* Seleccioné el valor del campo password en el cuerpo JSON.
    b. *Attack Type:* Sniper.
3. **Carga de Payloads:** Configuré una lista de 50 contraseñas comunes (ej. "123456", "password", "gym123").
4. **Ejecución:** Inicié el ataque automatizado sin configurar retardos (delays) entre peticiones.

2.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** Tras un número pequeño de intentos fallidos (ej. 5 o 10), la API debería responder con 429 Too Many Requests o un mensaje indicando que la cuenta ha sido bloqueada temporalmente.
- **Resultado Obtenido:**

Figura 28:

Captura de solicitud para login

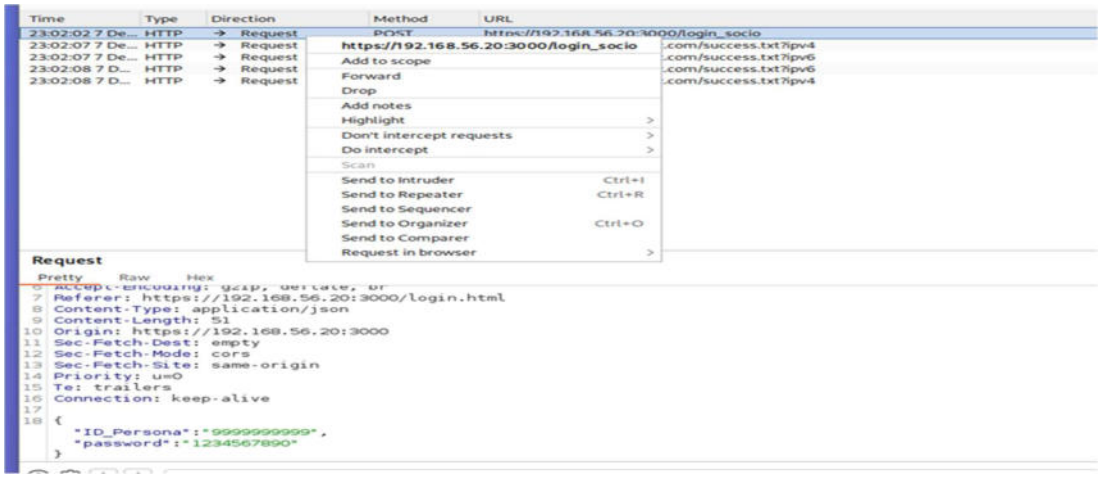
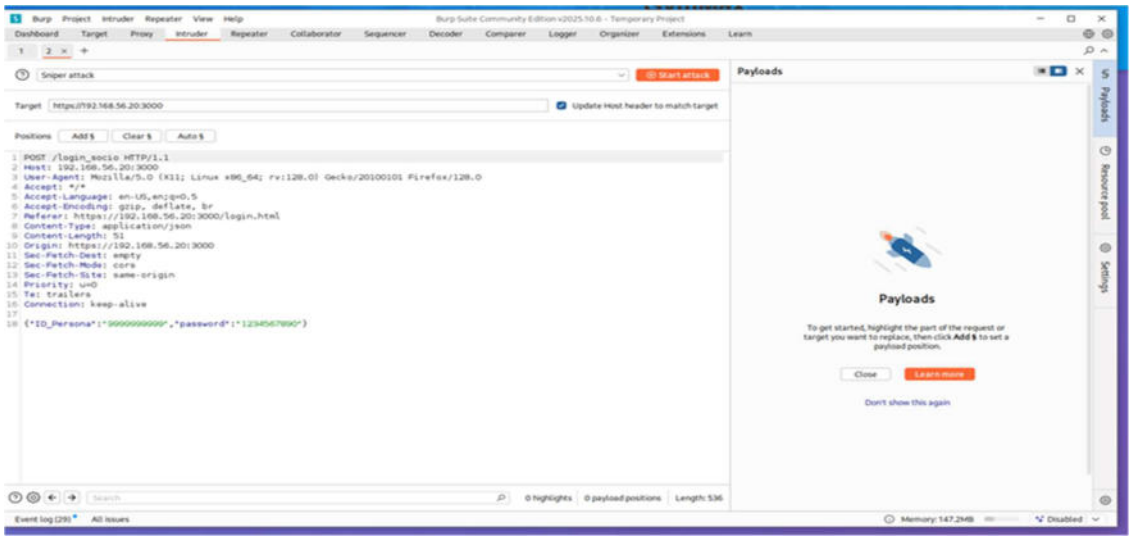


Figura 29:

Envio a Intruder



1. **Enviar a Intruder:**
    a. Haz clic derecho sobre la petición capturada en el Proxy.
    b. Selecciona **Send to Intruder** (o presiona Ctrl + I).
    c. La pestaña **Intruder** (arriba) se iluminará en naranja. Ve a ella.
2. **Configurar Posiciones (Positions):**
    a. Dentro de Intruder, ve a la sub-pestaña **Positions**.
    b. Verás la petición con varios símbolos § rodeando algunos datos. Burp intenta adivinar qué quieres atacar, pero suele marcar demasiadas cosas.
    c. Haz clic en el botón **Clear §** (a la derecha) para limpiar todo.
    d. Ahora, selecciona con el ratón **solo el valor** de la contraseña (la palabra prueba123).
    e. Haz clic en el botón **Add §**.
    f. Debería quedar así: "password": "§prueba123§".
    g. Asegúrate de que **Attack type** (arriba) esté en **Sniper** (por defecto).

3. **Configurar Payloads (Payloads):**
    a. Ve a la sub-pestaña **Payloads** (al lado de Positions).
    b. En **Payload settings [Simple list]**, aquí es donde pondrás las contraseñas que quieres probar.
    c. Puedes escribirlas una a una en la caja "Enter a new item" y darle a **Add**, o pegar una lista.

Figura 30:

Configuracion de Payloads

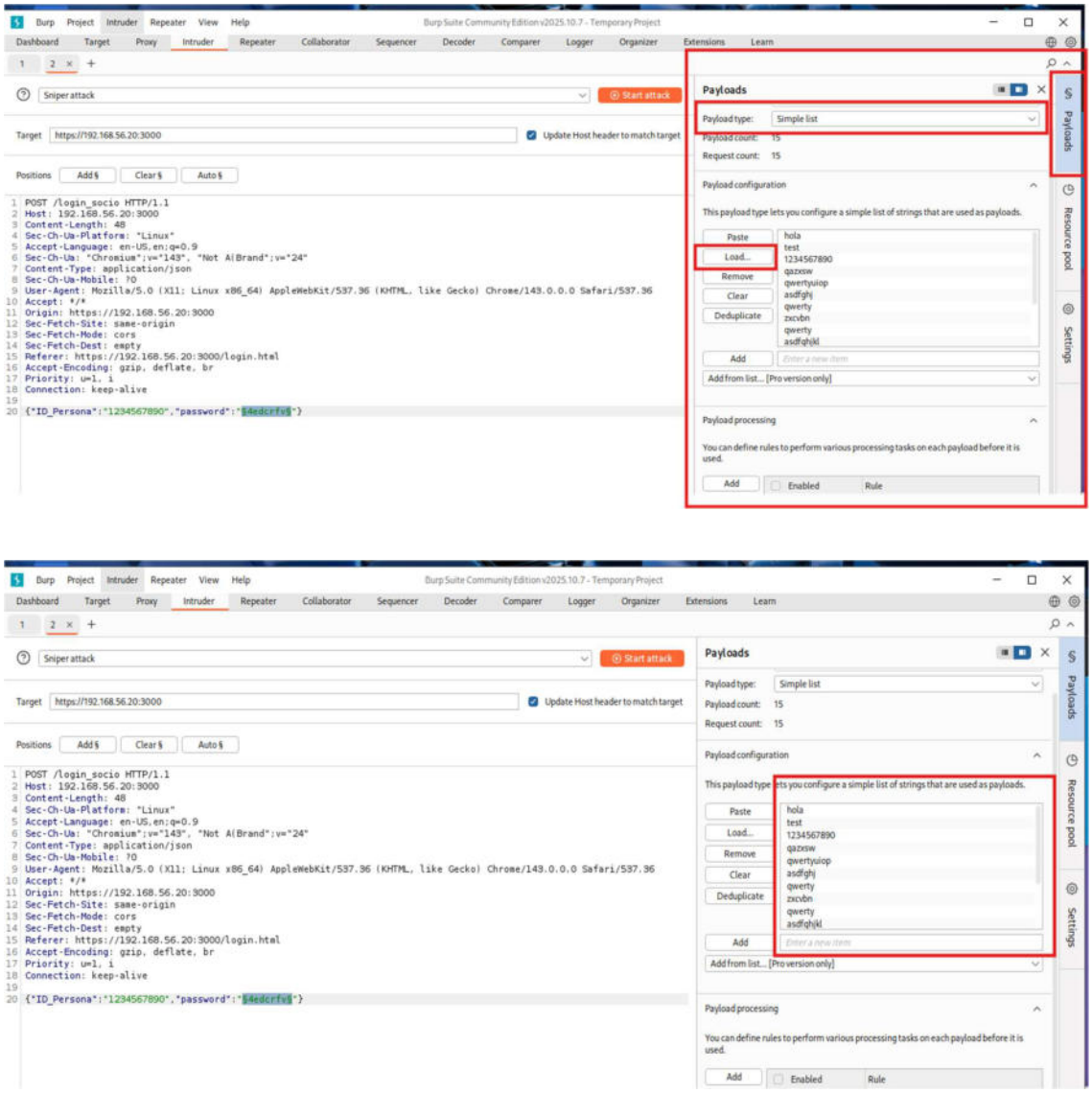


Figura 31:

Inicio de ataque

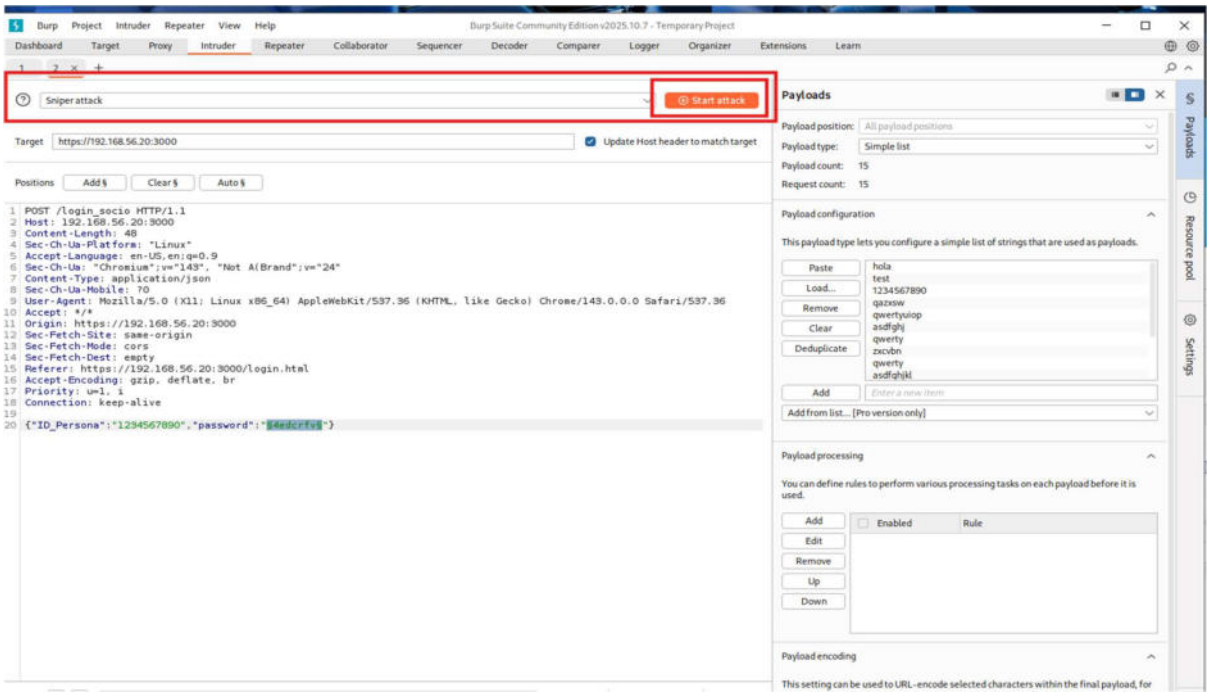
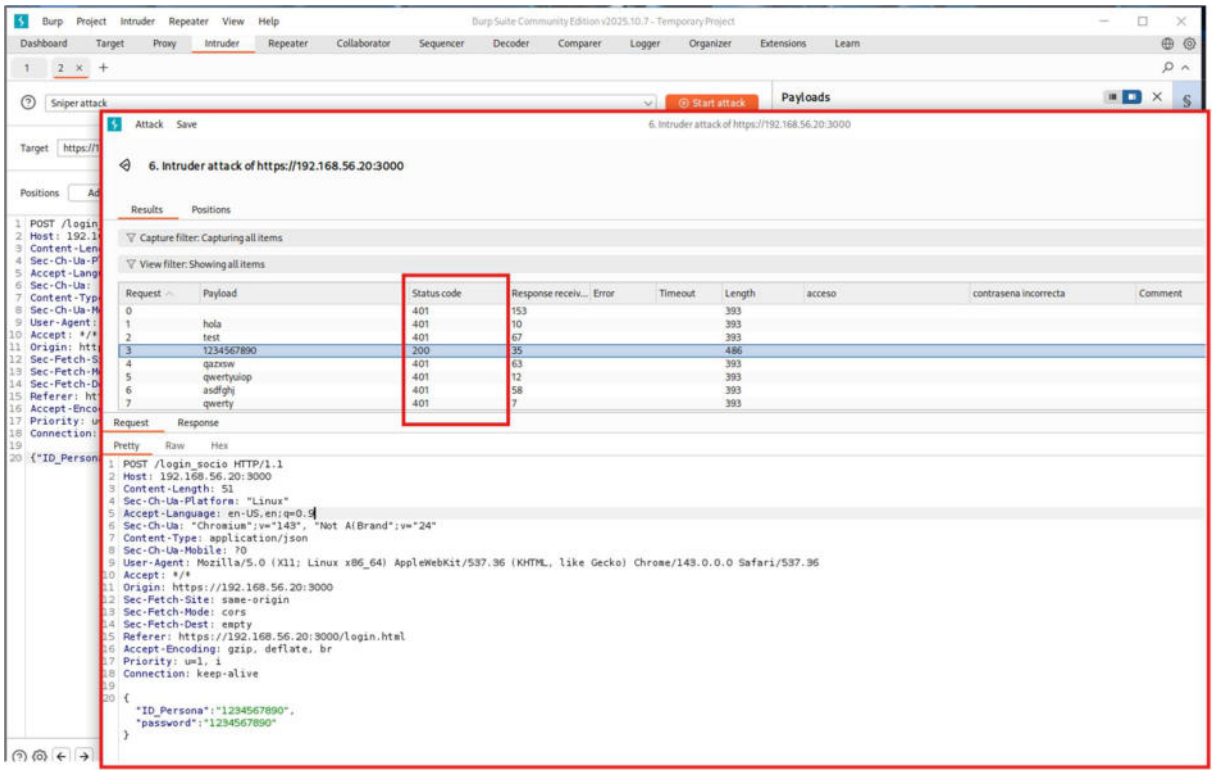
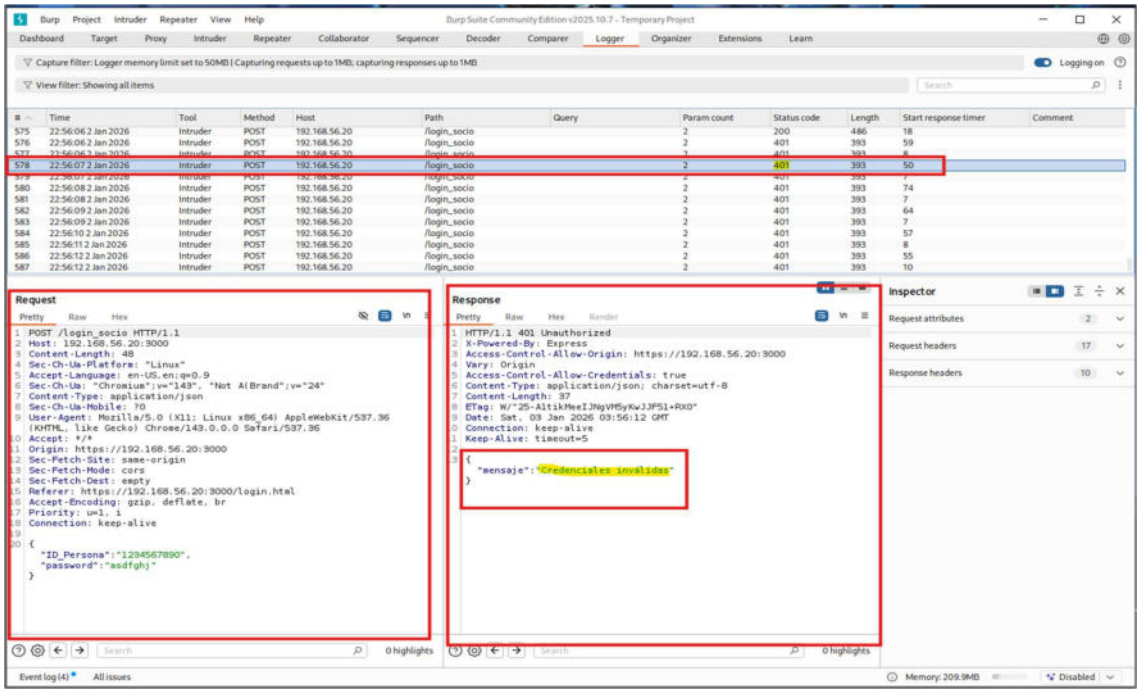


Figura 32:

Respuesta del de ataque





Al ejecutar el ataque de fuerza bruta, se observa que el servidor responde consistentemente con un código de estado 401, 200 y 500 Internal Server Error para todas las credenciales inválidas.

1. **Ausencia de Rate Limiting:** A pesar de los múltiples intentos fallidos y errores de servidor, la API continuó aceptando peticiones sin bloquear la IP o el usuario (no se observan códigos 429).
2. **Manejo de Errores Deficiente:** El retorno de códigos 500 sugiere que las excepciones no están siendo capturadas correctamente (try-catch) en la lógica de autenticación, lo cual podría derivar en una denegación de servicio (DoS) si el error consume muchos recursos

Figura 33:

Respuesta del de ataque

3. Pruebas de Autorización a Nivel de Propiedad del Objeto (BOPLA)

Referencia: API3:2023 - Broken Object Property Level Authorization (BOPLA / Mass Assignment)

3.1. Objetivo de la Prueba

Comprobar si el servidor permite a un usuario modificar propiedades de objetos que no deberían ser modificables por ese rol o nivel de permiso (p. ej., un usuario estándar

modificando su propio rol, balance o estado de cuenta). Esto se conoce como *Mass Assignment* o Asignación Masiva.

3.2. Escenario de Prueba

Un usuario con rol "Empleado" intentará consumir un endpoint diseñado exclusivamente cambiar el estado del cliente.

3.3. Procedimiento Técnico

1. **Inicio de Sesión:** En el navegador, iniciar sesión con el **Usuario A** (el usuario de bajo privilegio).

2. **Actualización Normal:** Realiza una acción de actualización de perfil o de un recurso permitido (ej. cambiar tu nombre o dirección).

3. **Captura la Solicitud:** Burp Suite capturará la solicitud PUT o POST del *endpoint* de actualización.

3.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** El servidor debe denegar la acción con 403 Forbidden al detectar que el rol asociado a la sesión no es "Admin/Empleado".
- **Resultado Obtenido:** El servidor respondió 200 OK

Figura 34:

Preparando búsqueda del empleado

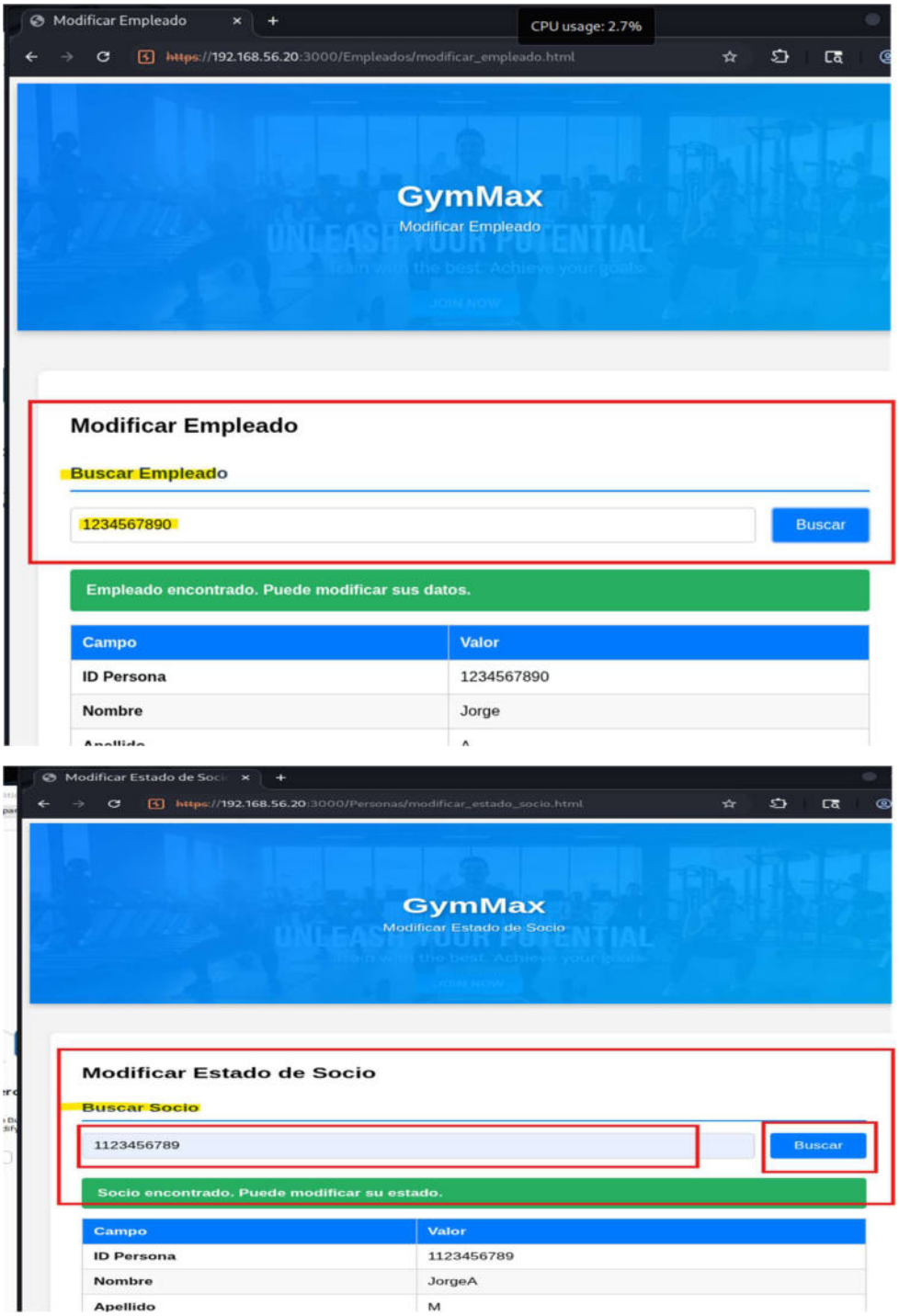


Figura 35:

Actualización normal

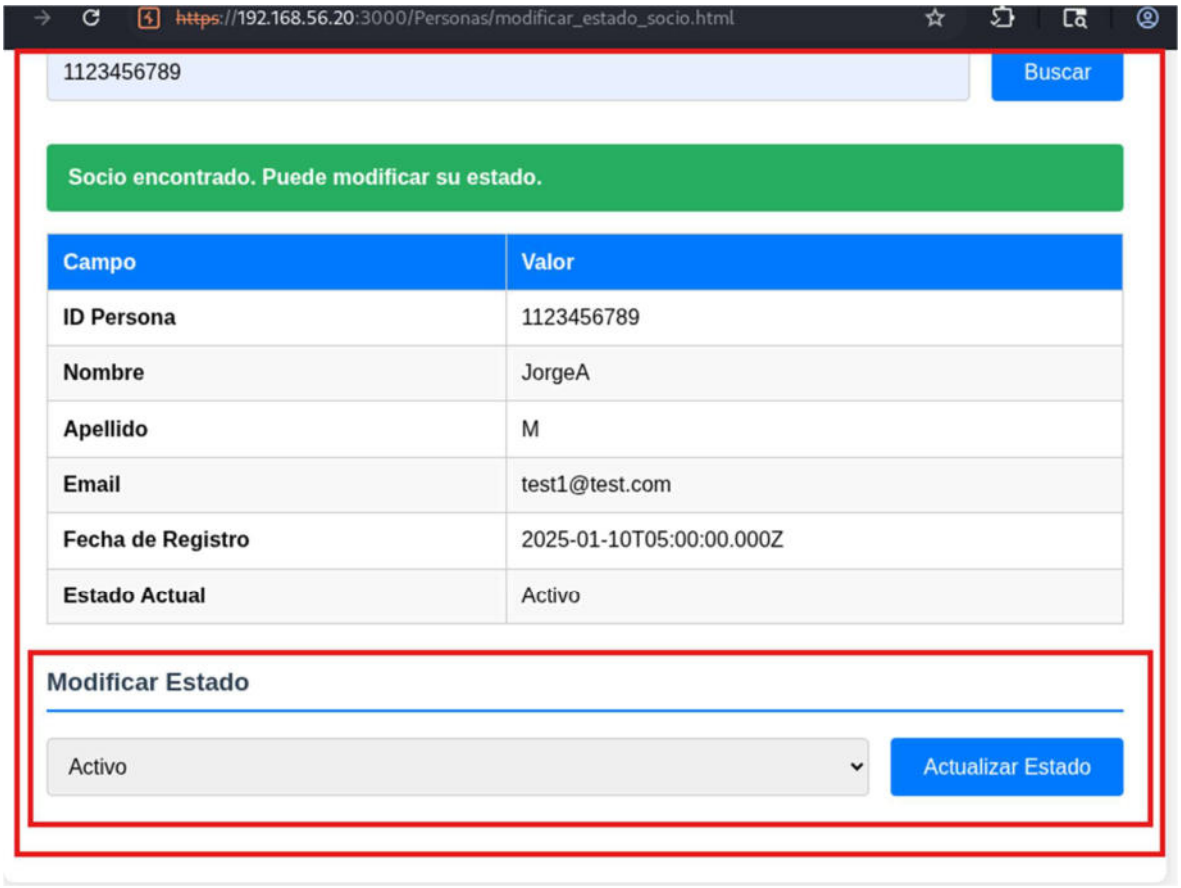
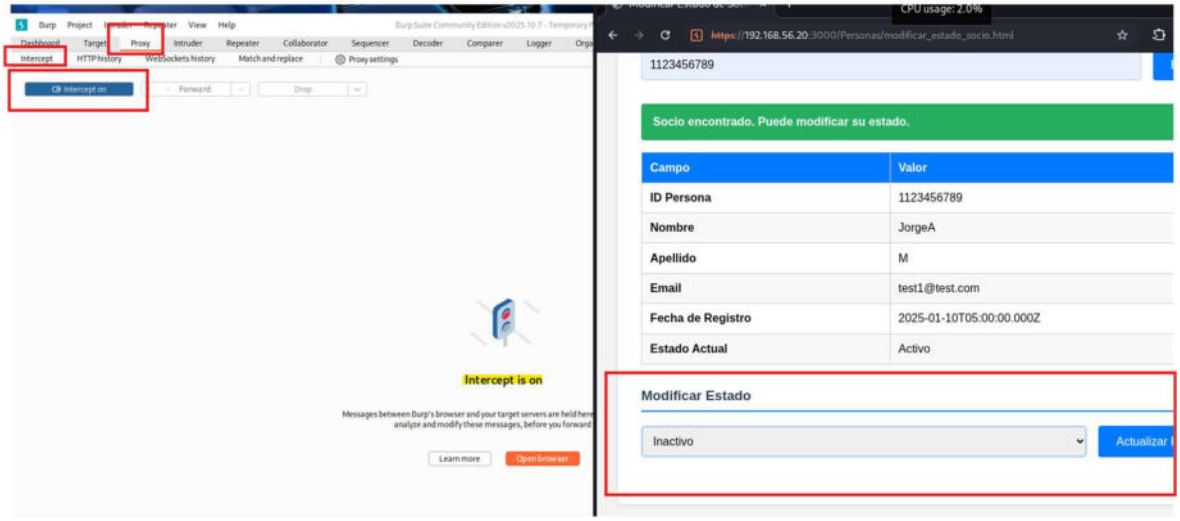


Figura 36:

Captura la solicitud en burpsuite



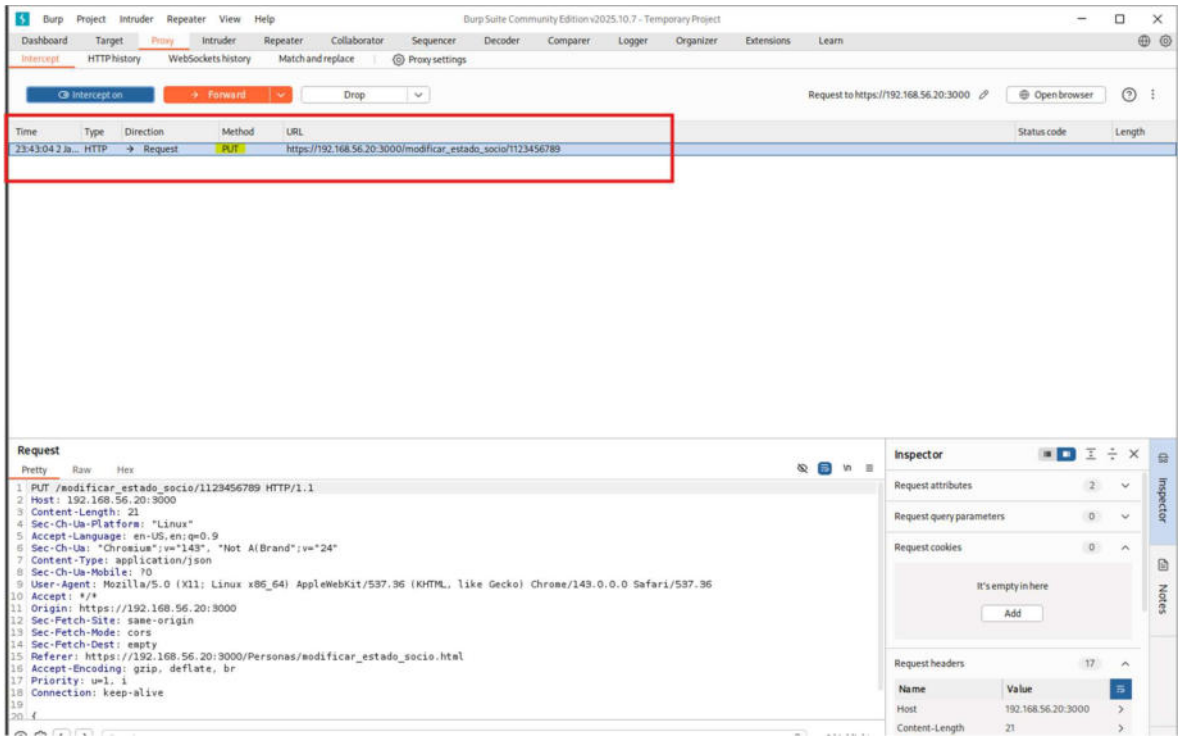


Figura 37:

Capturado el método, lo enviamos a "Repeater"

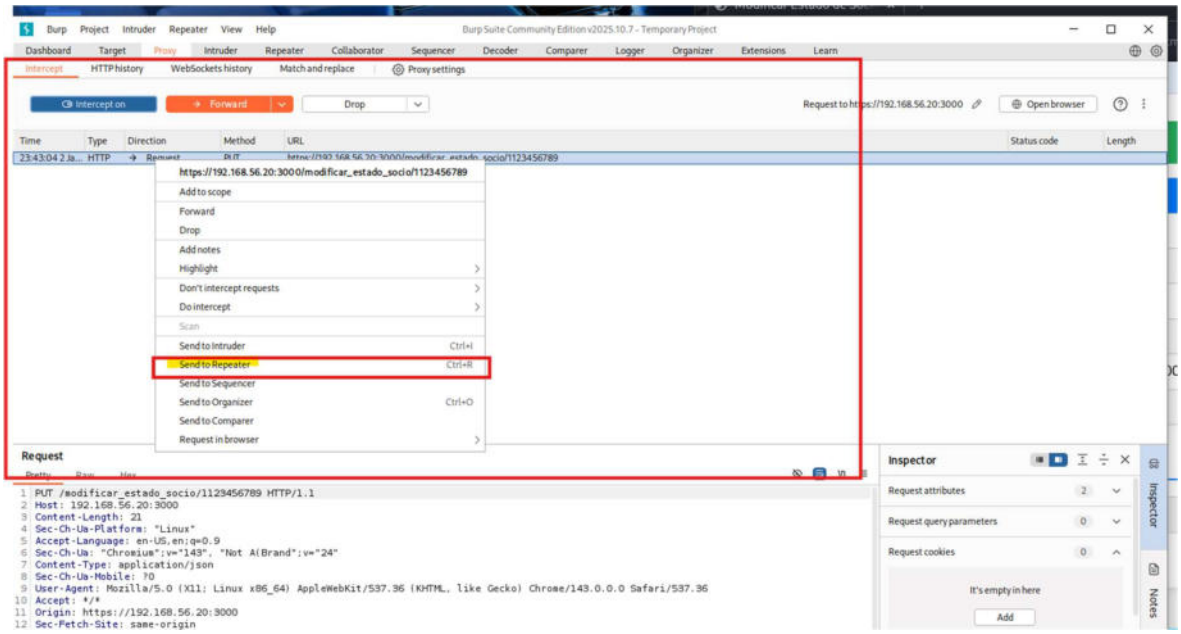


Figura 38:

Apagamos el Intercept y nos dirigimos a la pestaña de "Repeater"

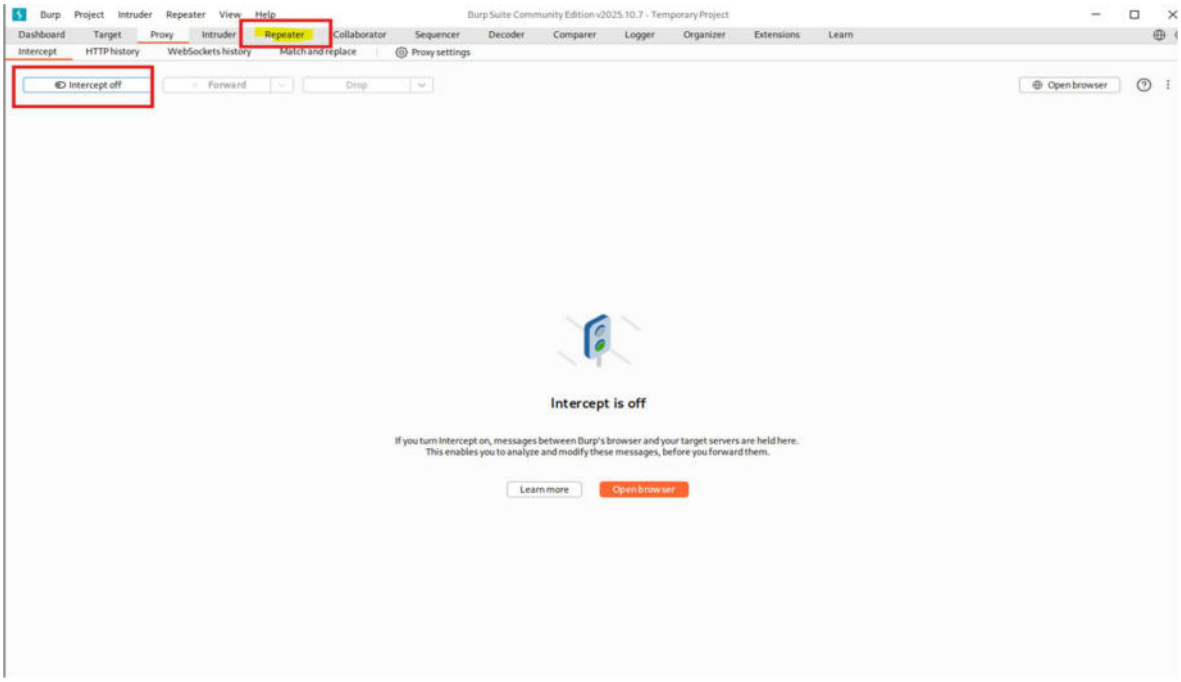
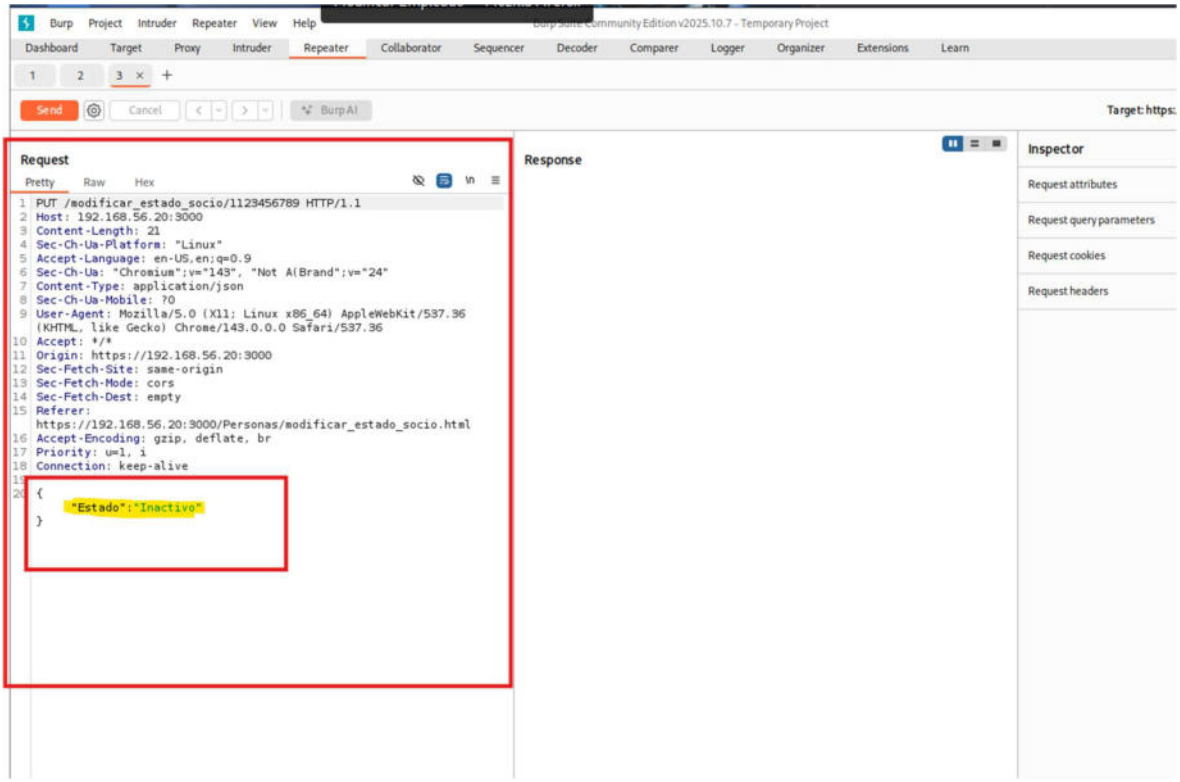


Figura 39:
Analizamos y cambiamos variable para enviar



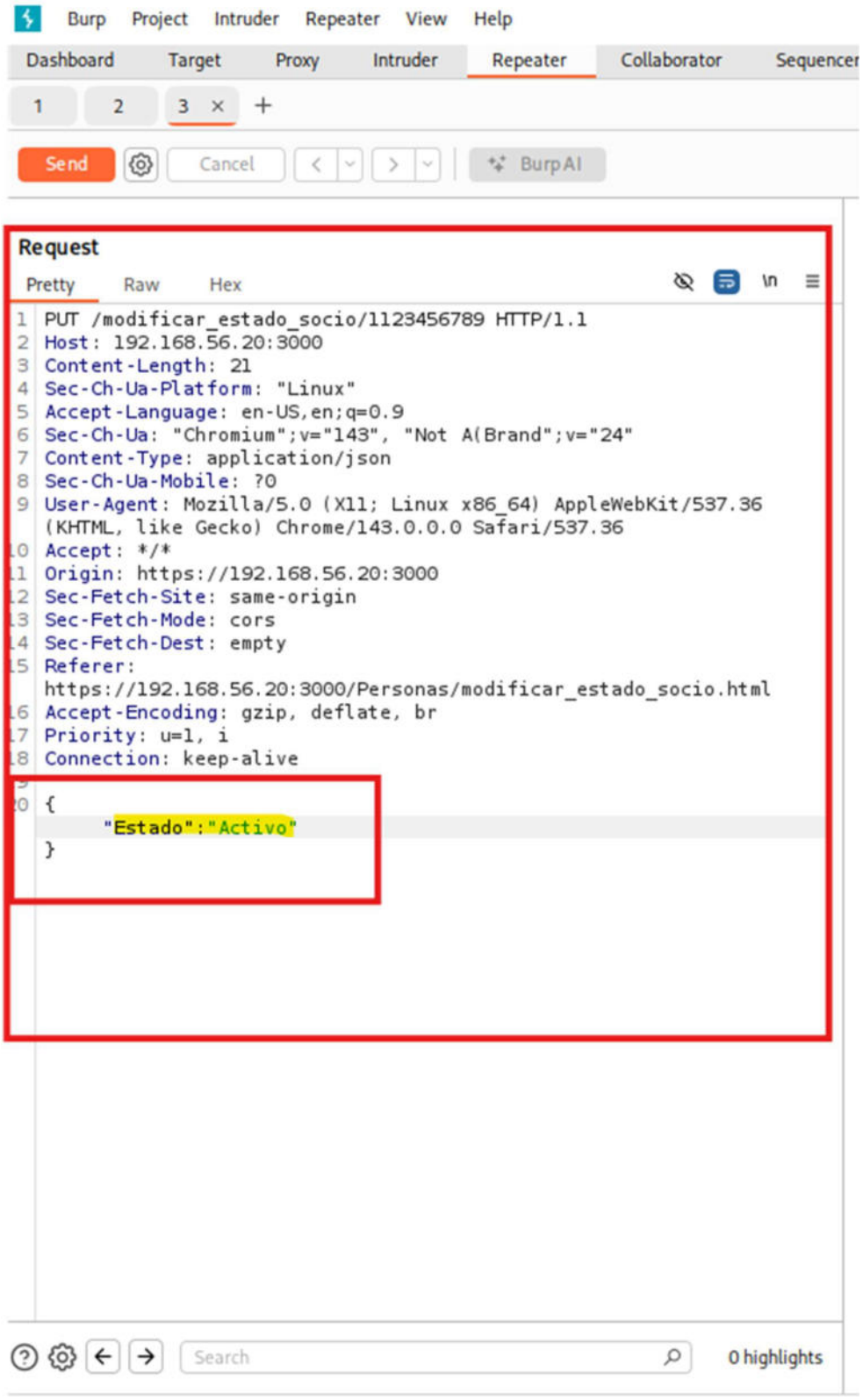
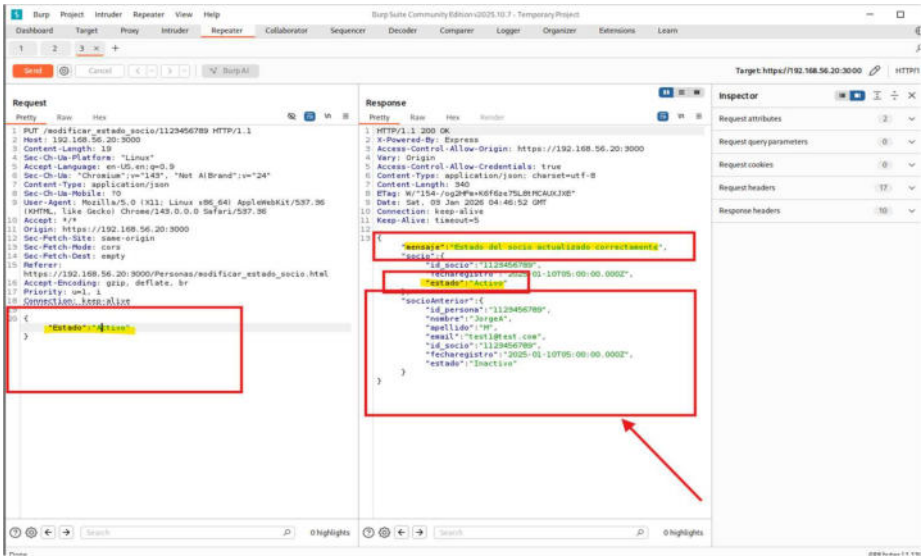
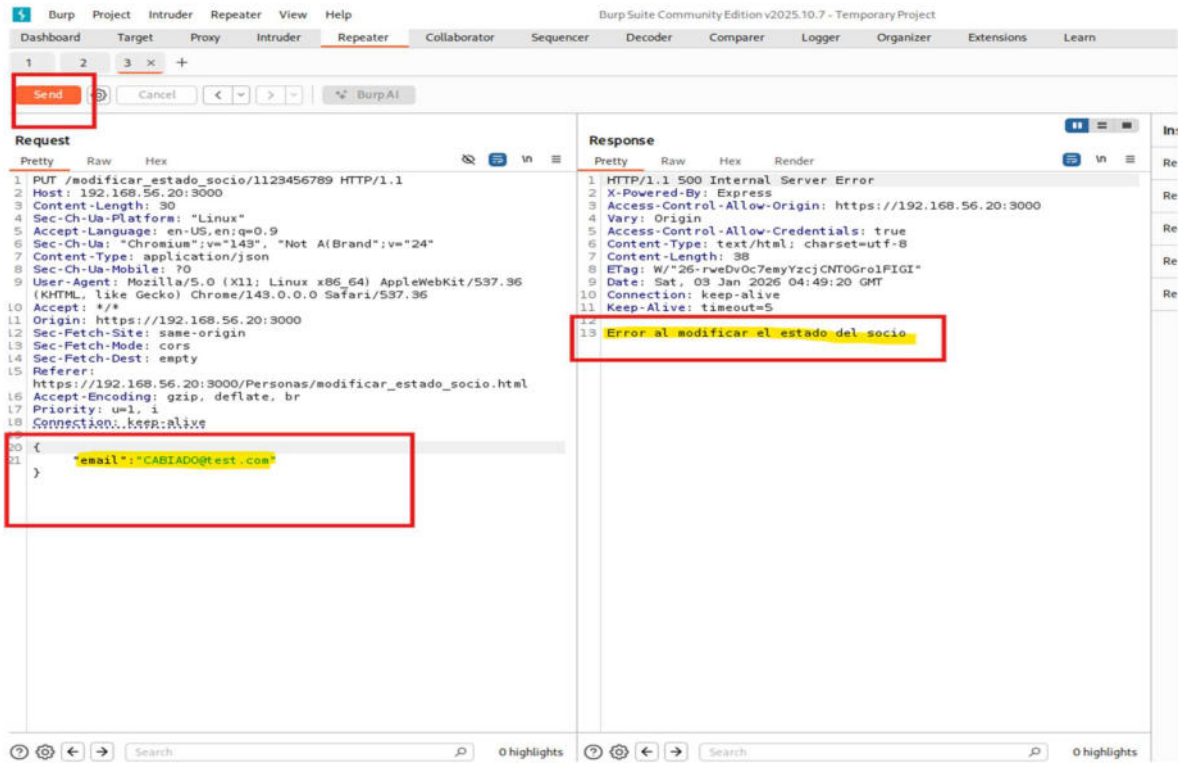

Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer
1 2 3 ×
Send Cancel Burp AI
Request
Pretty Raw Hex
1 PUT /modificar_estado_socio/1123456789 HTTP/1.1
2 Host: 192.168.56.20:3000
3 Content-Length: 21
4 Sec-Ch-Ua-Platform: "Linux"
5 Accept-Language: en-US,en;q=0.9
6 Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand";v="24"
7 Content-Type: application/json
8 Sec-Ch-Ua-Mobile: ?0
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
10 Accept: */*
11 Origin: https://192.168.56.20:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://192.168.56.20:3000/Personas/modificar_estado_socio.html
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=1, i
18 Connection: keep-alive
{
"Estado":"Activo"
}
Search
0 highlights

Figura 40:
Realizamos el cambio y enviamos la simulación.



Ahora vamos a realizar otro tipo de inyección con propiedades prohibidas o no permitidas para conocer el resultado.

Figura 41:
Realizamos el cambio y enviamos la simulación.



Confirmamos que en primera instancia actualizo el campo, devolviendo el código 200 OK, en cambio en el segundo intento de actualización genera el error y devolviendo el código 500.

4. Consumo Ilimitado

Referencia: API4:2023 - Unrestricted Resource Consumption (Consumo de Recursos)

4.1. Objetivo de la Prueba

Verificar la presencia y efectividad de los mecanismos de Rate Limiting, Throttling y validación de límites de tamaño (*payload*) para prevenir la degradación del servicio o el fallo total debido a peticiones excesivas o sobredimensionadas.

4.2. Escenario de Prueba

Un usuario con rol "Socio" intentará consumir un endpoint diseñado exclusivamente para el rol "Empleado/Admin" para eliminar un registro.

4.3. Procedimiento Técnico

1. En el navegador, ejecutar una solicitud al *endpoint* de **alto consumo** (ej. la generación de un reporte).

2. Capturar la solicitud (GET o POST) en el **Proxy** de Burp.

3. Hacer clic derecho sobre la solicitud y seleccionar **"Send to Intruder"**.

4.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** Si todas las peticiones regresan 200 OK y el tiempo de respuesta promedio aumenta drásticamente

- **Resultado Obtenido:** No se cae el servicio, sin embargo, si hay un alto consumo en los recursos del servidor de API

Figura 42:
Realizamos la consulta de máquinas, en el menú principal.

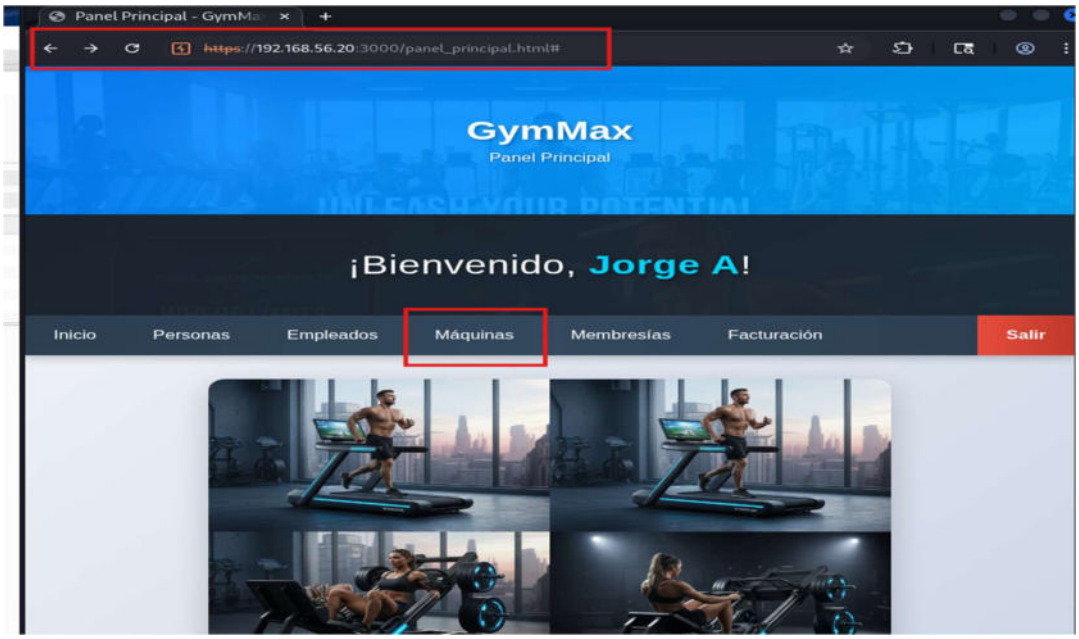


Figura 43:
Cargamos la consulta.

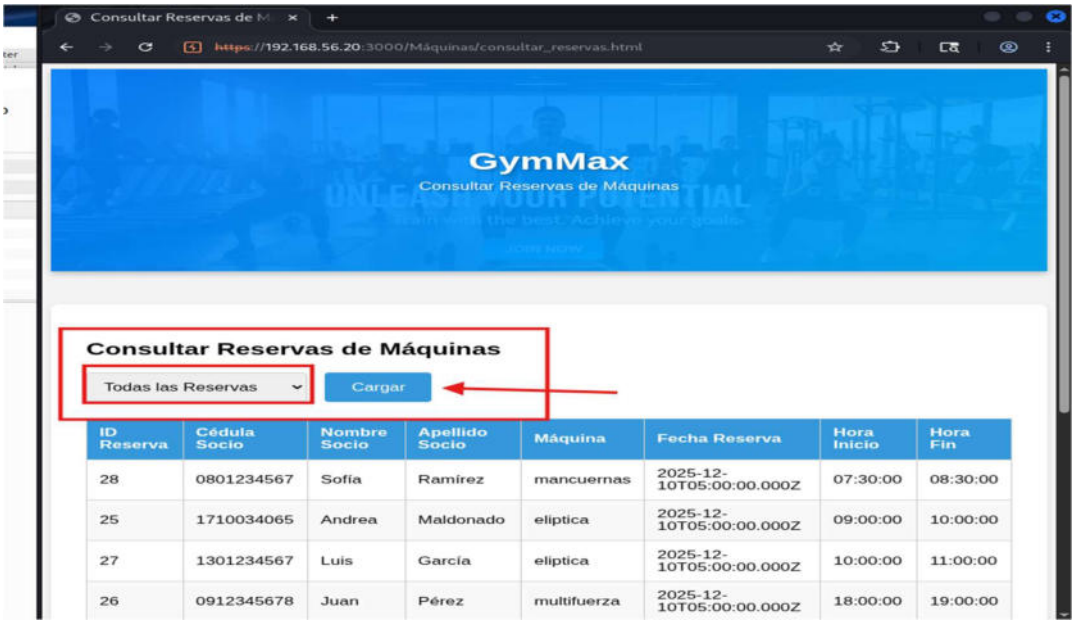


Figura 44:
Capturamos la solicitud GET con Burpsuite.

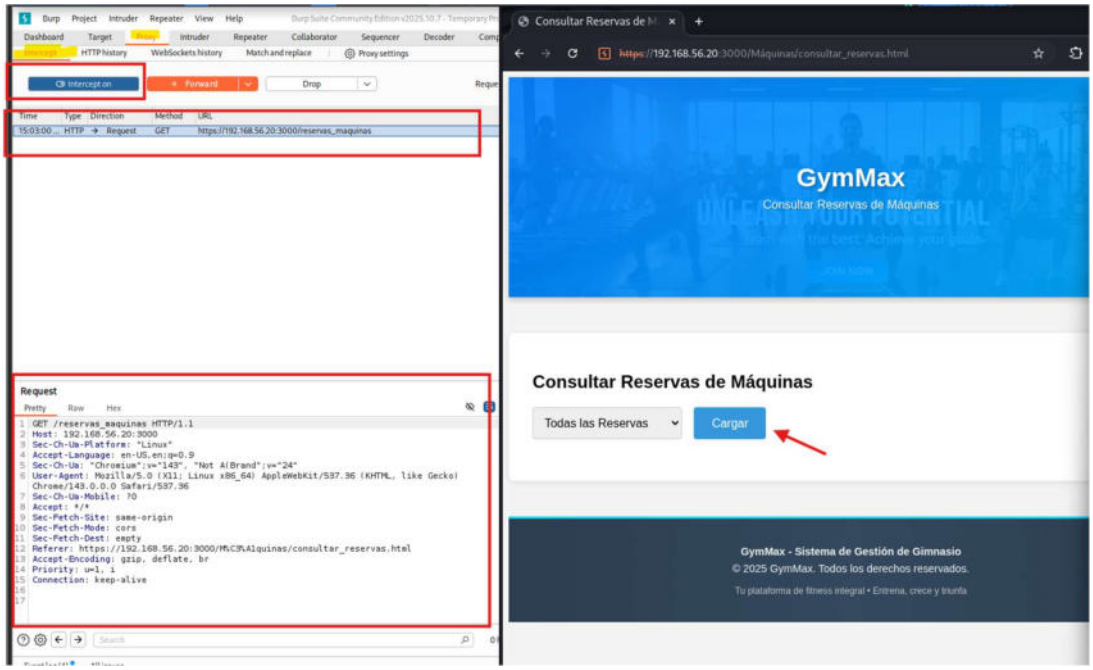


Figura 45:
Enviamos lo capturado a “Intruder”.

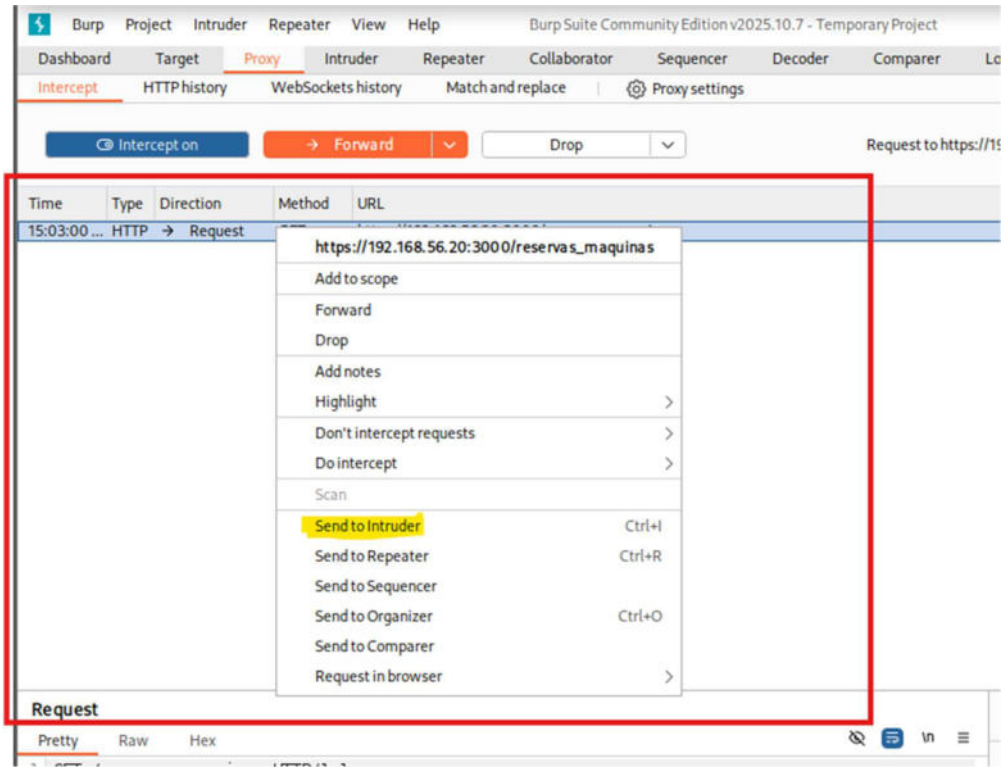
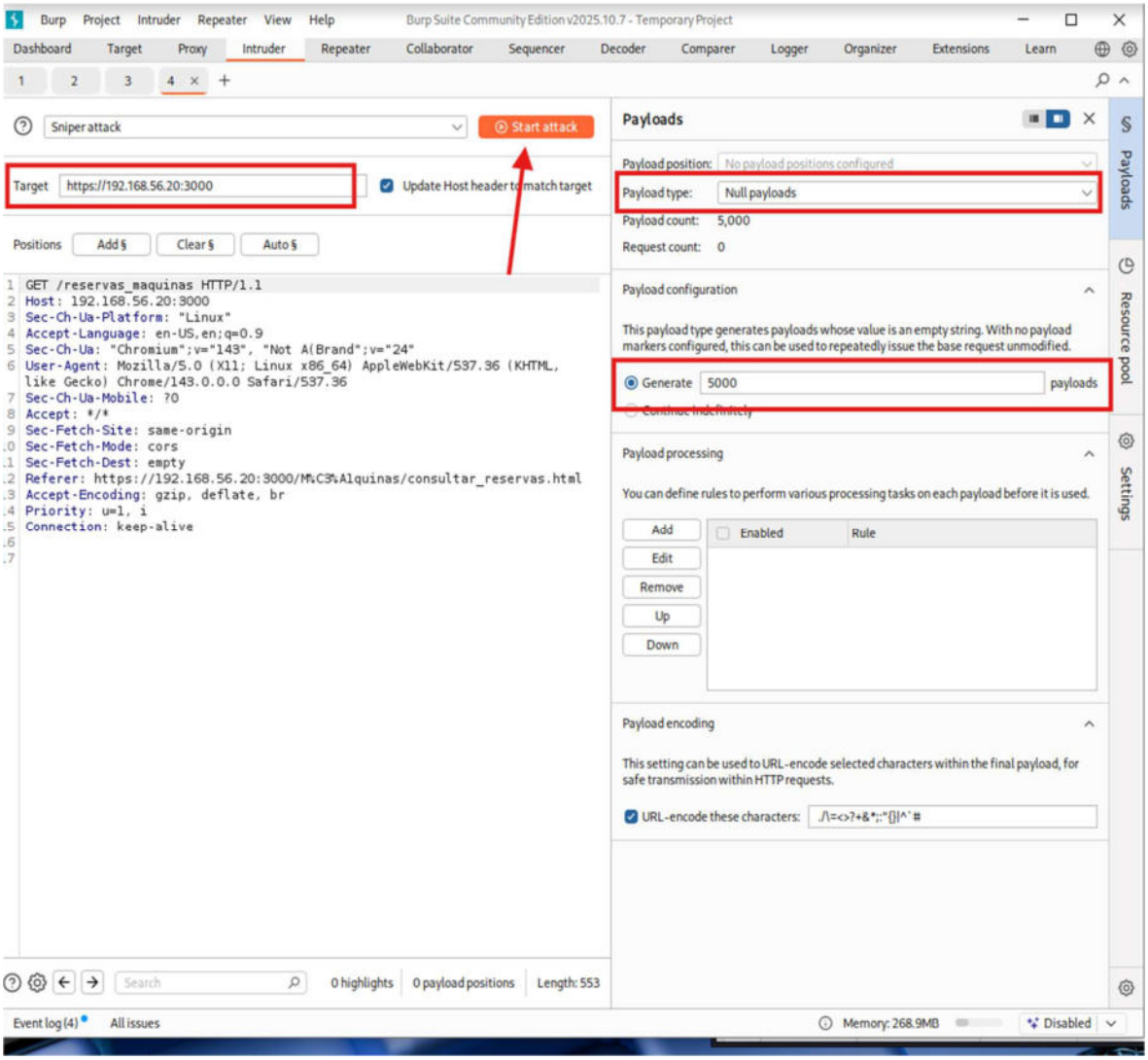


Figura 46:
Configuración para el ataque.



En **Payloads**, vamos a configurar los siguientes parámetros para luego lanzar el ataque en:

a. P**ayload Type:** Seleccionar "Null Payloads".

b. **Payload configuration:** Establecer el número de peticiones a 5,000.

Figura 47:
Evidencia del ataque.

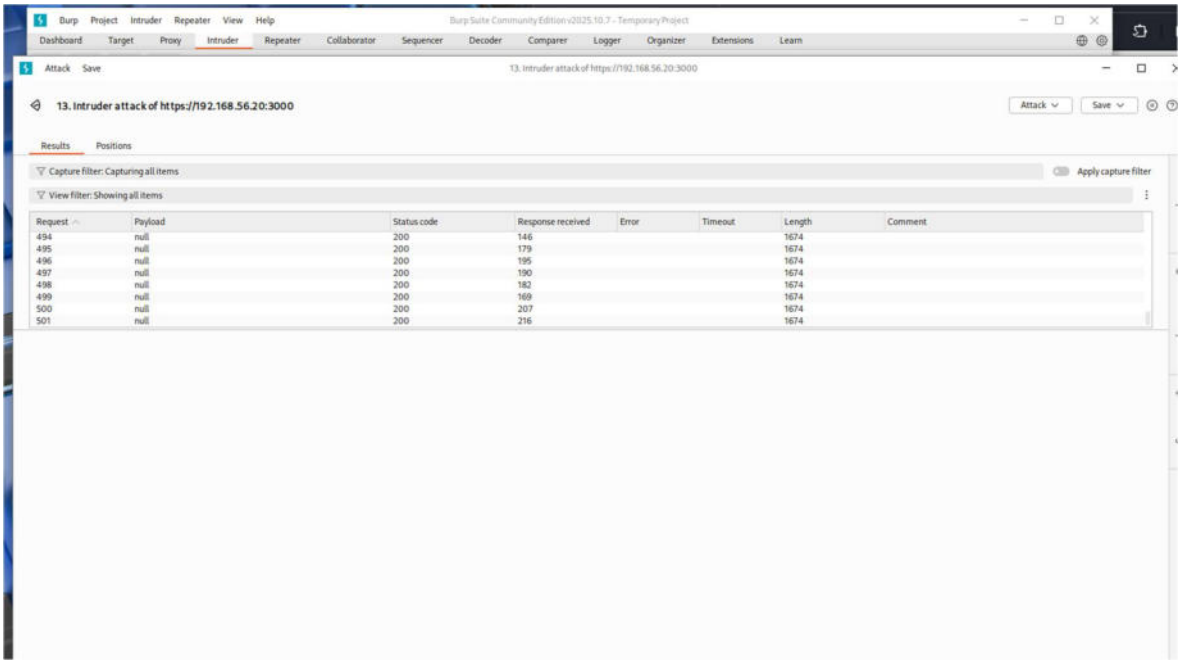


Figura 48:
Evidencia del ataque.

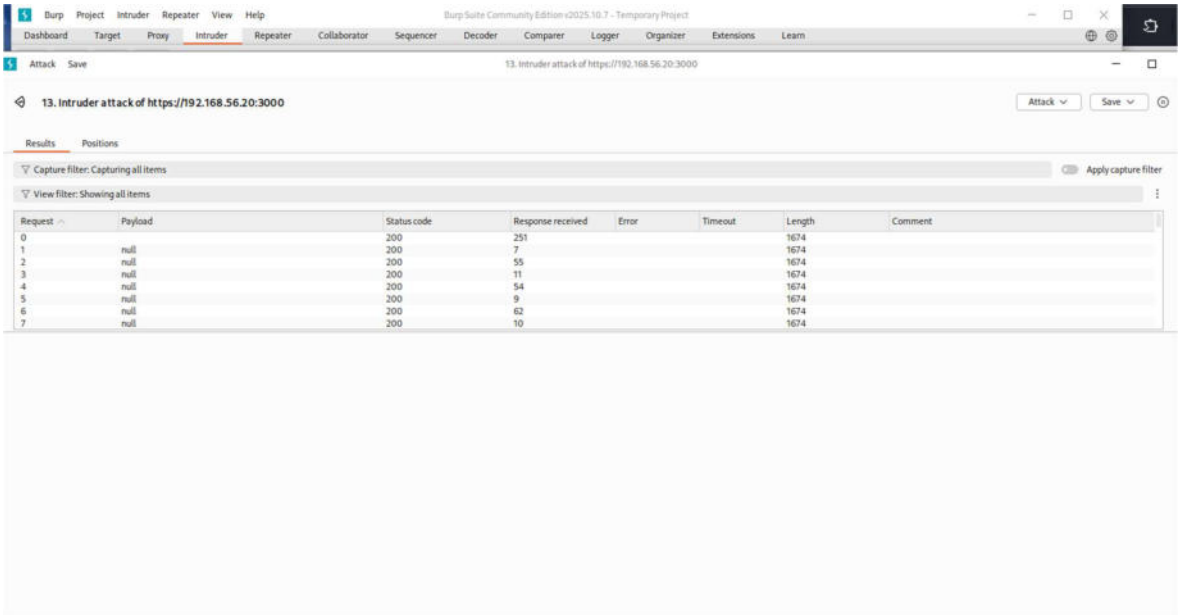
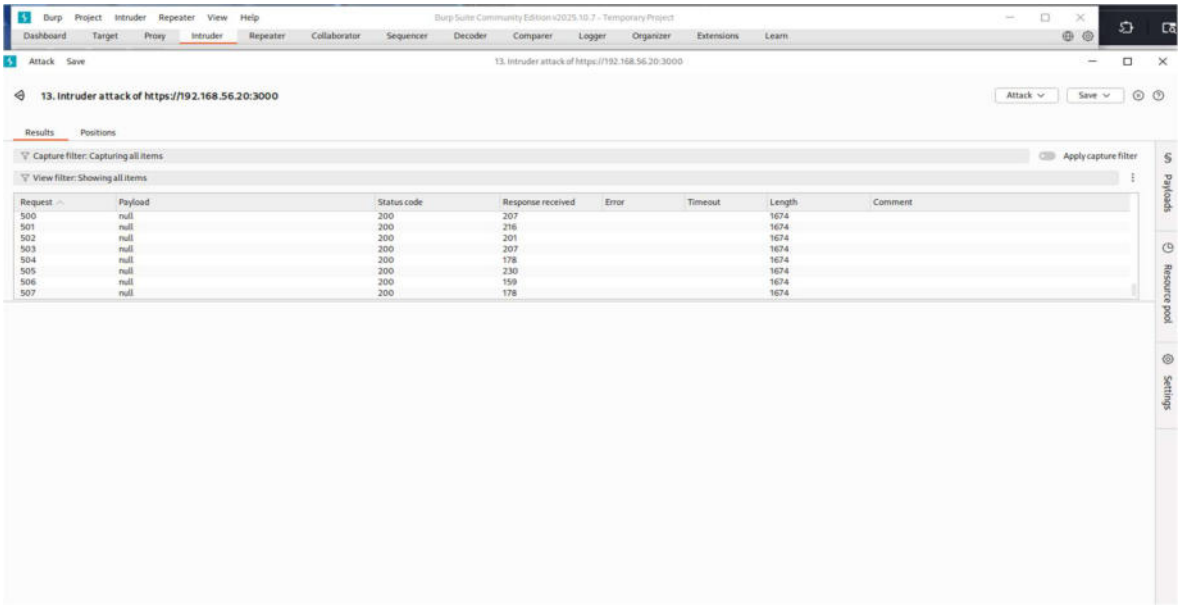


Figura 49:
Evidencia del ataque.



Todas las peticiones regresan 200 OK y el tiempo de respuesta promedio aumenta drásticamente

Figura 50:
Consumo de recursos del servidor del servicio API.

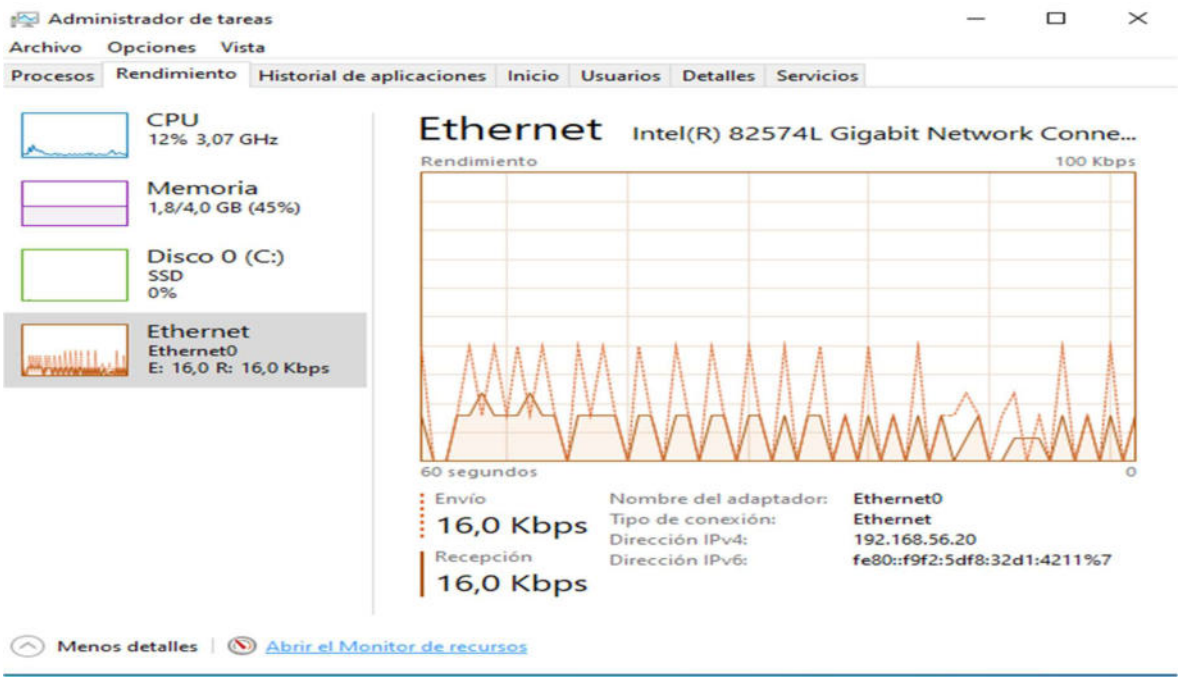
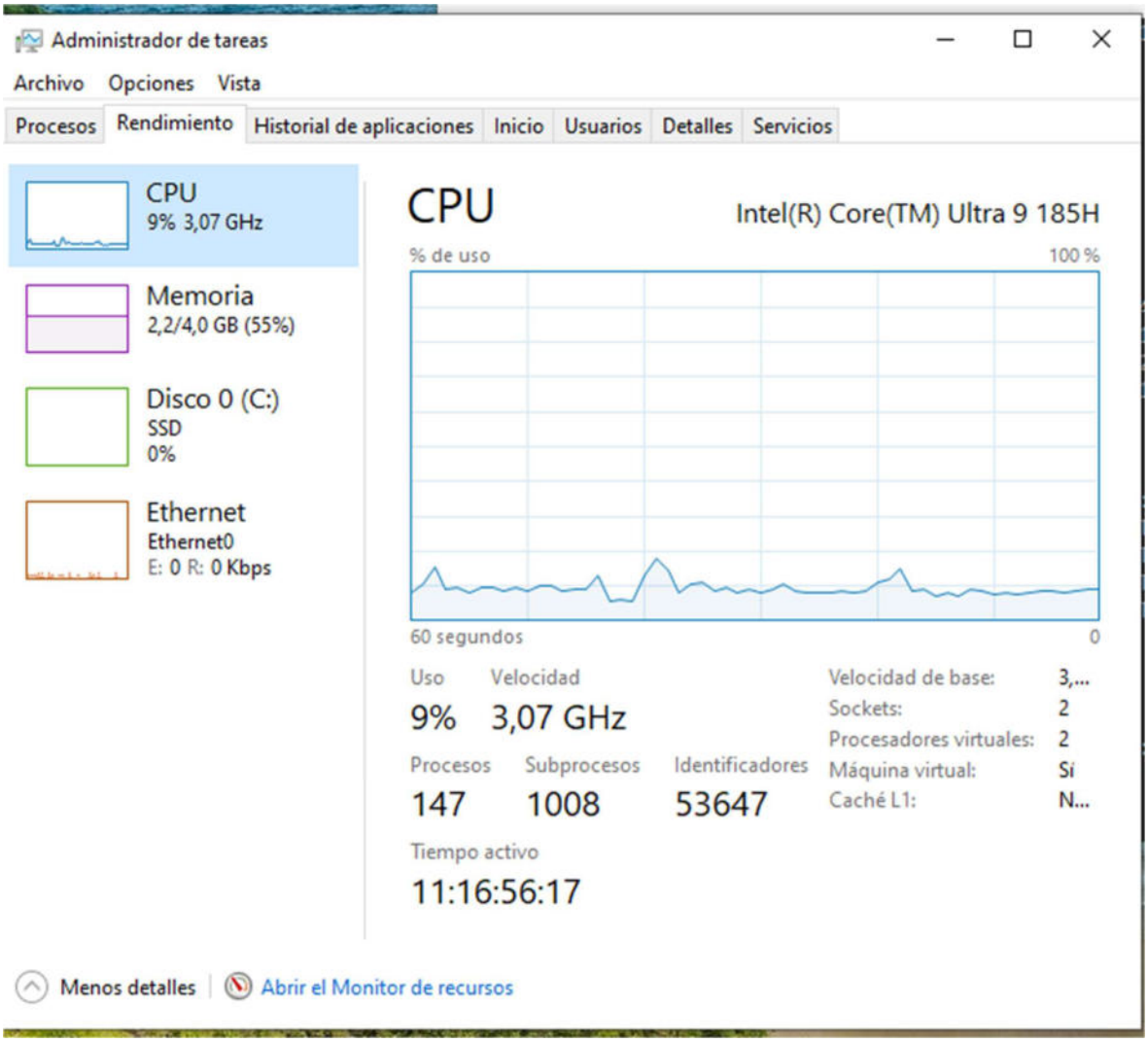


Figura 51:
Consumo de recursos del servidor del base de datos.



5. Pruebas de Autorización a Nivel de Función (BFLA)

Referencia: API5:2023 - Broken Function Level Authorization

5.1. Objetivo de la Prueba

Comprobar que un usuario con privilegios bajos (Usuario Estándar) no puede acceder, ejecutar o descubrir endpoints destinados exclusivamente a usuarios con privilegios altos (Administrador o Super-Usuario).

5.2. Escenario de Prueba

Un usuario con rol "Socio" intentará consumir un endpoint diseñado exclusivamente para el rol "Empleado/Admin" para eliminar un registro.

5.3. Procedimiento Técnico

1. **Reconocimiento:** Identifiqué mediante la documentación Swagger el endpoint administrativo GET.
2. **Sesión:** Me autentiqué en el sistema con un usuario de bajo privilegio (Socio).
3. **Construcción de Ataque:** Dado que el frontend del Socio no tiene botón "Agregar/Actualizar/Eliminar", utilicé **Repeater** para construir la petición manualmente:
    a. *Método:* GET
    b. *URL:* https://192.168.56.20:3000/panel_secundario.html
4. **Ejecución:** Envié la solicitud contra un ID de prueba existente.

5.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** El servidor debe denegar la acción con 403 Forbidden al detectar que el rol asociado a la sesión no es "Admin/Empleado".

- **Resultado Obtenido:** *El servidor respondió 200 OK y el usuario fue actualizado, evidenciando una falta de control de roles en el backend*

Figura 52:
Ingreso al aplicativo como cliente.

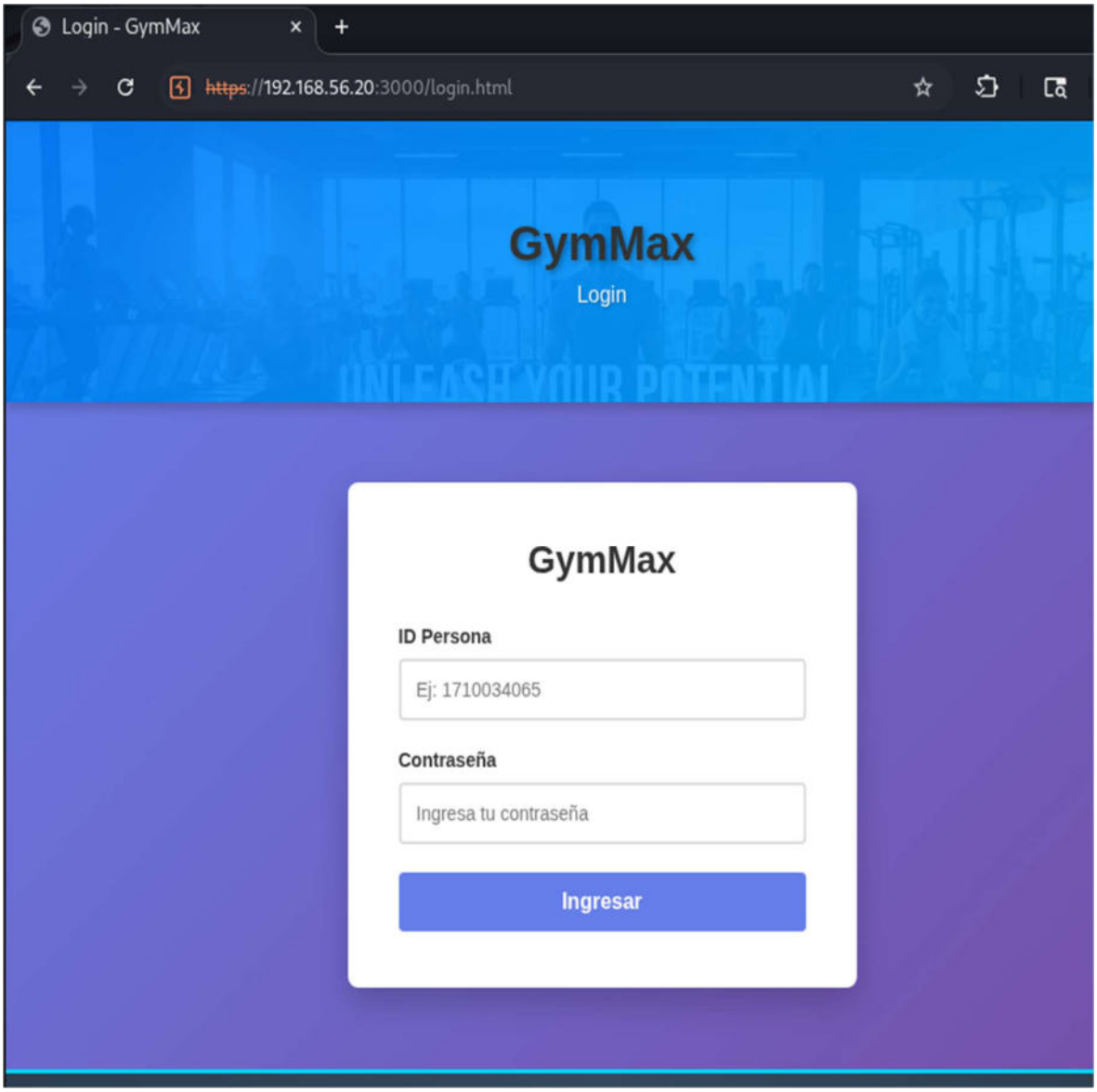


Figura 53:
Comprobación de ingreso.

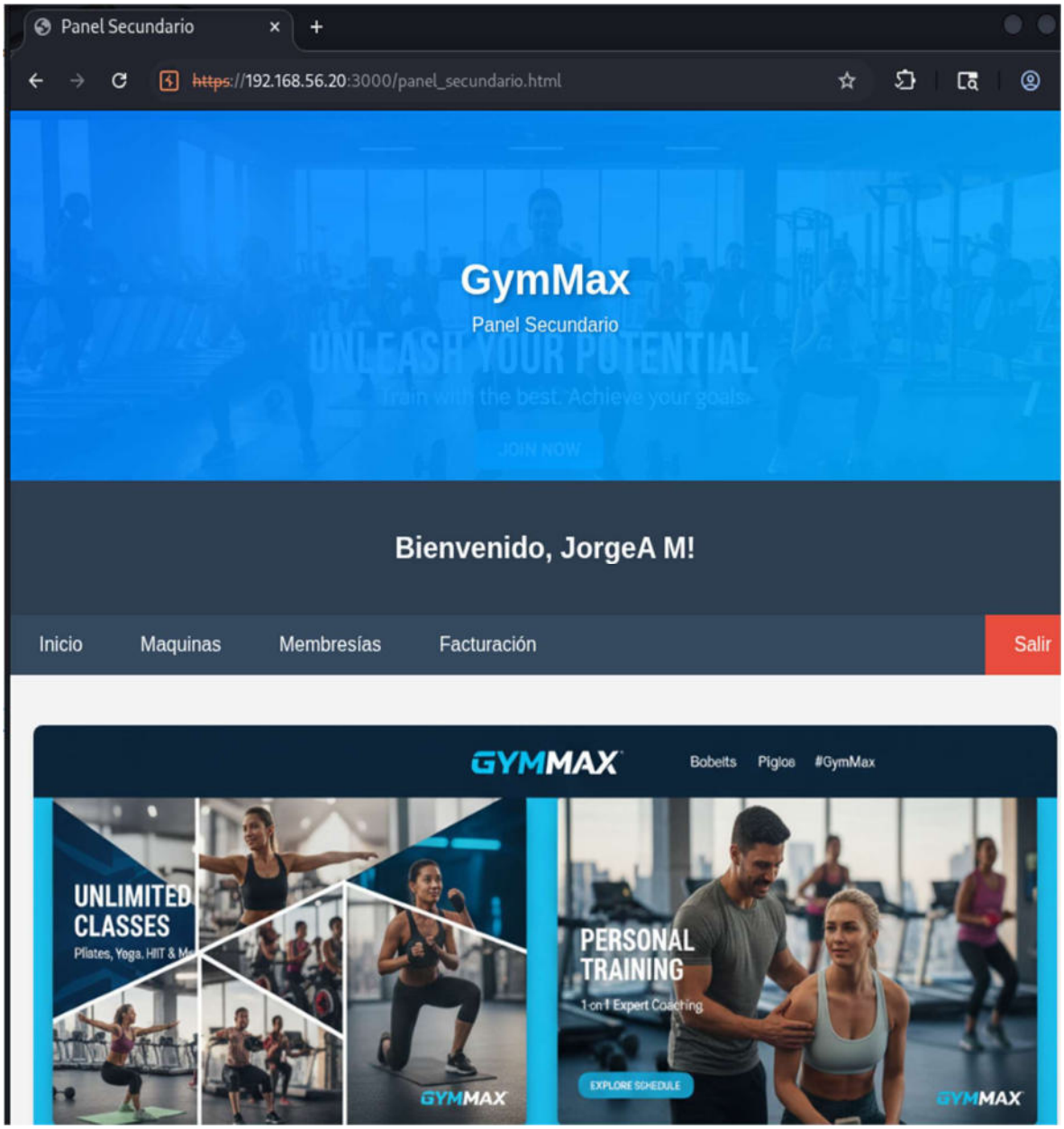


Figura 54:
Captura del método con Burpsuite.

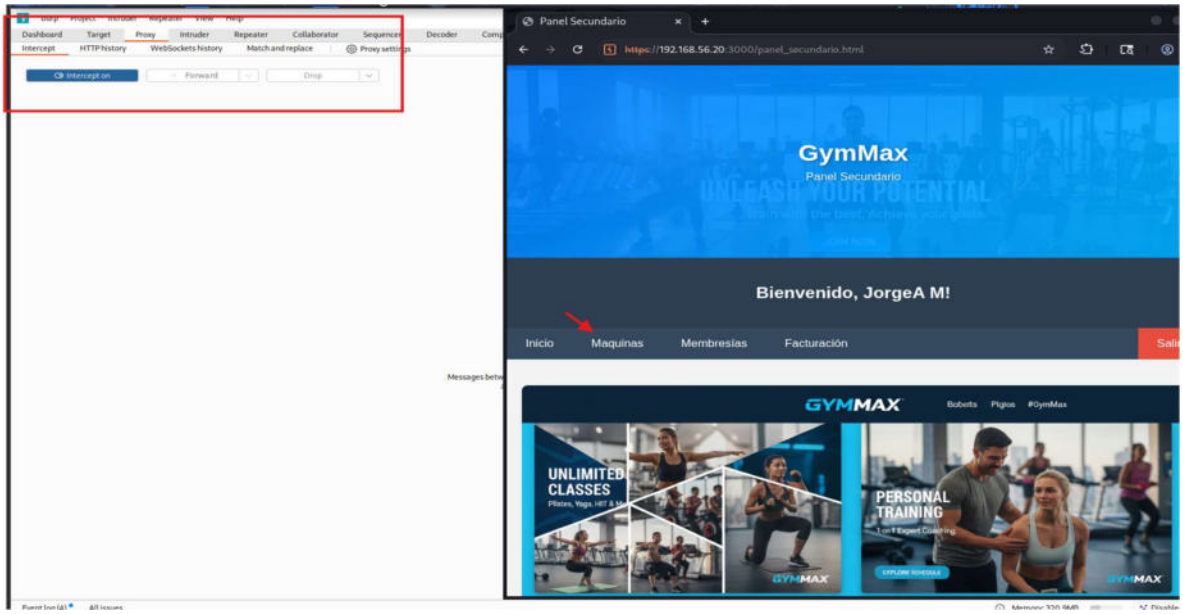


Figura 55:
Obtención del método.

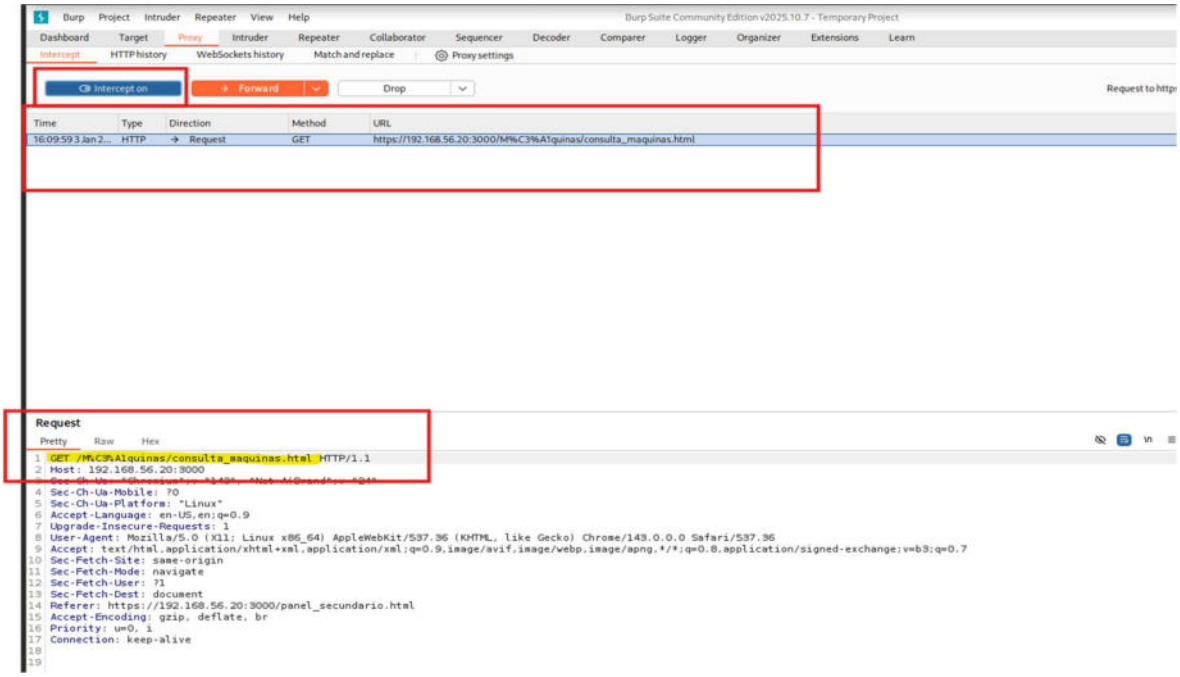


Figura 56:
Envió de la captura a “Repeater”.

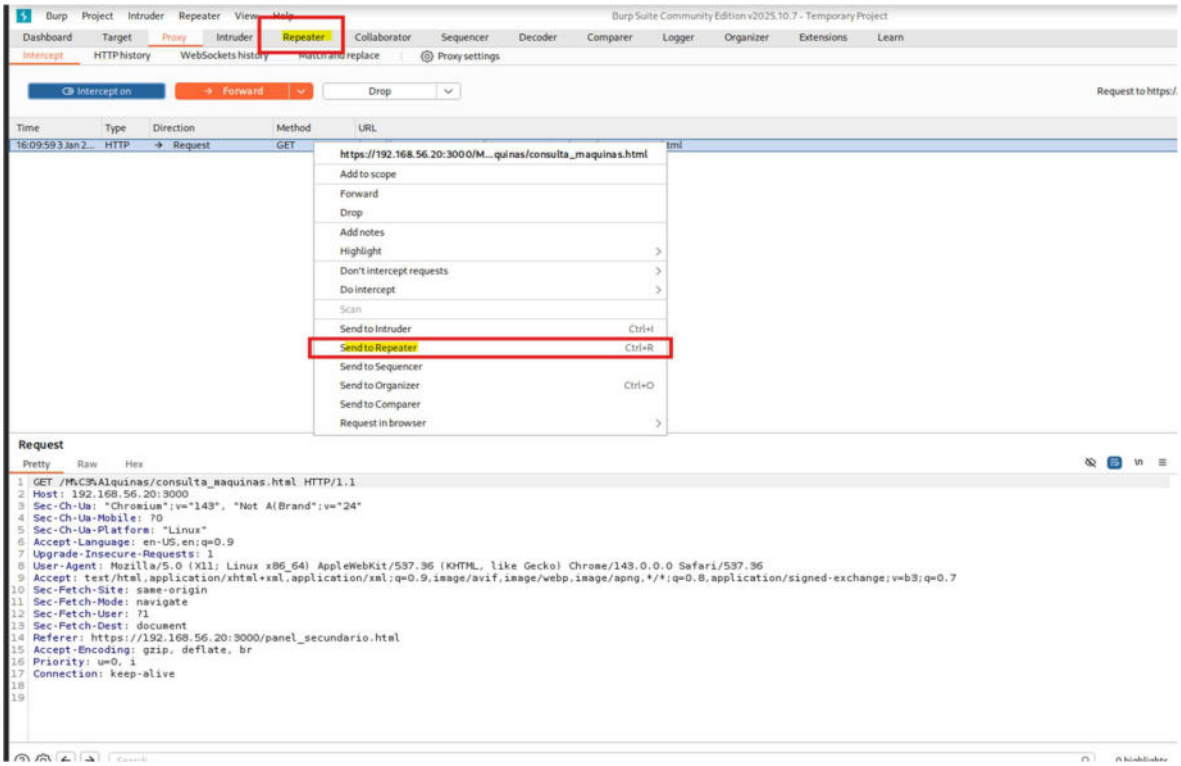
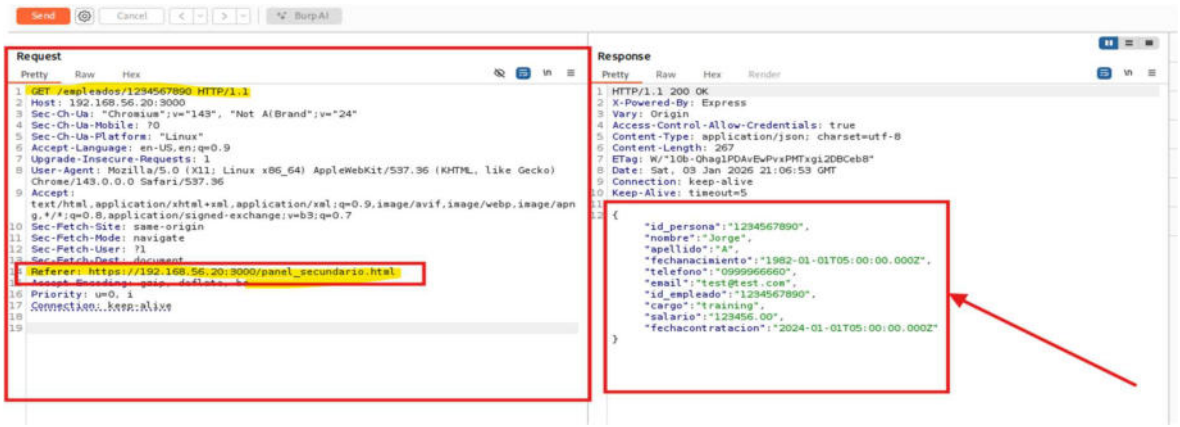
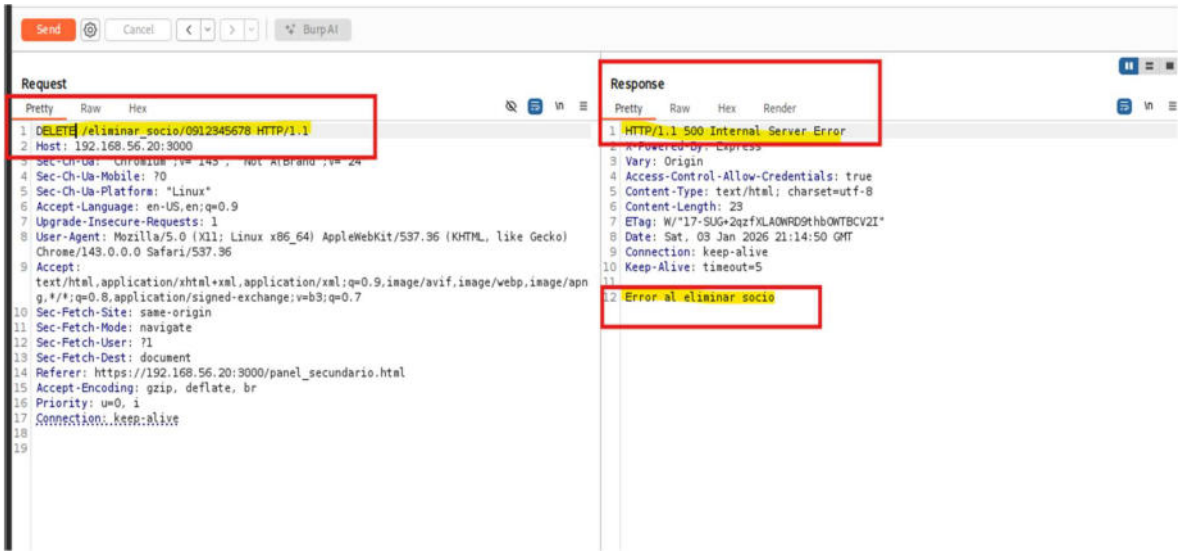


Figura 57:
Envido del método GET para la obtención de datos de los empleados desde perfil de socio.



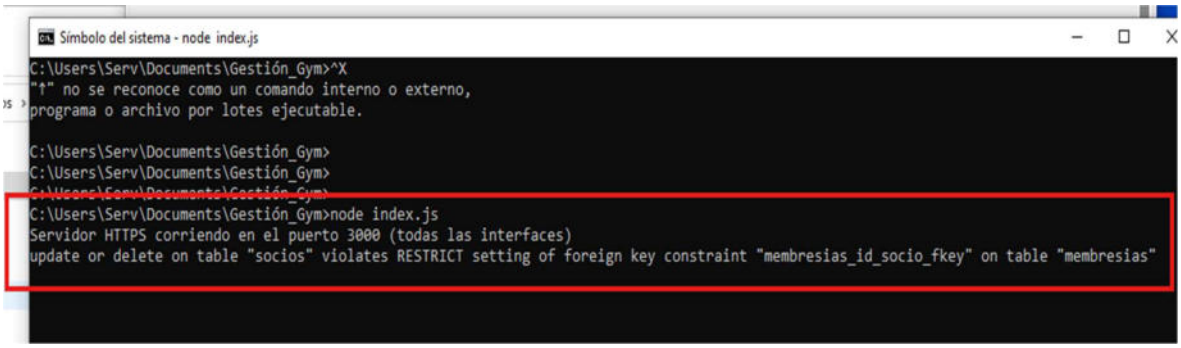
Mensaje 200 de proceso completado y correcto.

Figura 58:
Envió del método DELETE.



Mensaje de ERROR 500 que no proceso adecuadamente por tema de privilegios.

Figura 59:
Mensaje de erro en servidor de la ejecución de la API.



6. Pruebas de Lógica de Negocio (Manipulación de Pagos)

Referencia: API6:2023 - Unrestricted Access to Sensitive Business Flows

6.1. Objetivo de la Prueba

Verificar la integridad del flujo de compras, asegurando que el usuario no pueda alterar arbitrariamente el precio de las membresías antes de que se procese el pago.

6.2. Escenario de Prueba

Un usuario intentará adquirir una membresías, mediante la manipulación de la solicitud de pago.

6.3. Procedimiento Técnico

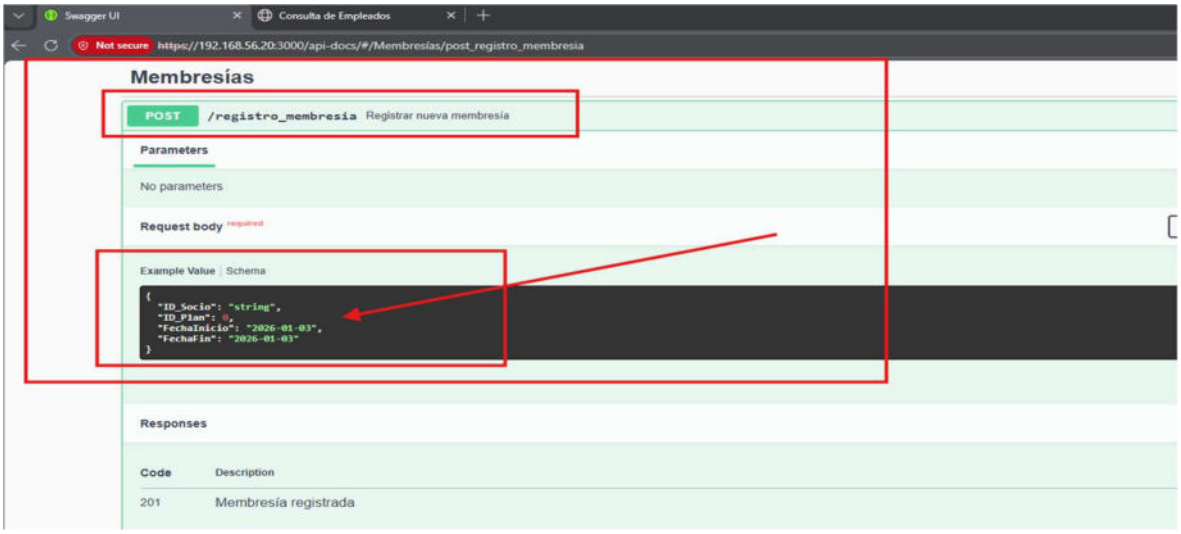
1. **Flujo Normal:** Inicié el proceso de compra en el navegador hasta llegar a la confirmación.
2. **Interceptación:** Capturé la petición final al endpoint POST /registraro_membresia.
3. **Manipulación:** En el **Repeater**, modifiqué el valor del campo ID_Plan.
4. **Ejecución:** Envié la solicitud manipulada.

6.4. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** La API debería validar en el backend que el monto enviado coincida con el precio real del plan asociado a la id_membresia, rechazando la transacción si hay discrepancia.
- **Resultado Obtenido:** La API aceptó el registro y activó la membresía, demostrando una vulnerabilidad crítica de lógica de negocio.

Ingreso manual desde el Swagger, para identificar el método e intentar realizar el ingreso manual desde Burpsuite.

Figura 60:
Revisión de endpoint del método POST.



En Burpsuite, nos dirigimos a la pestana “Repeater”, y armamos el código para el ingreso de datos, como se muestra en la figura 60.

Figura 61:
Revisión de endpoint del método POST.

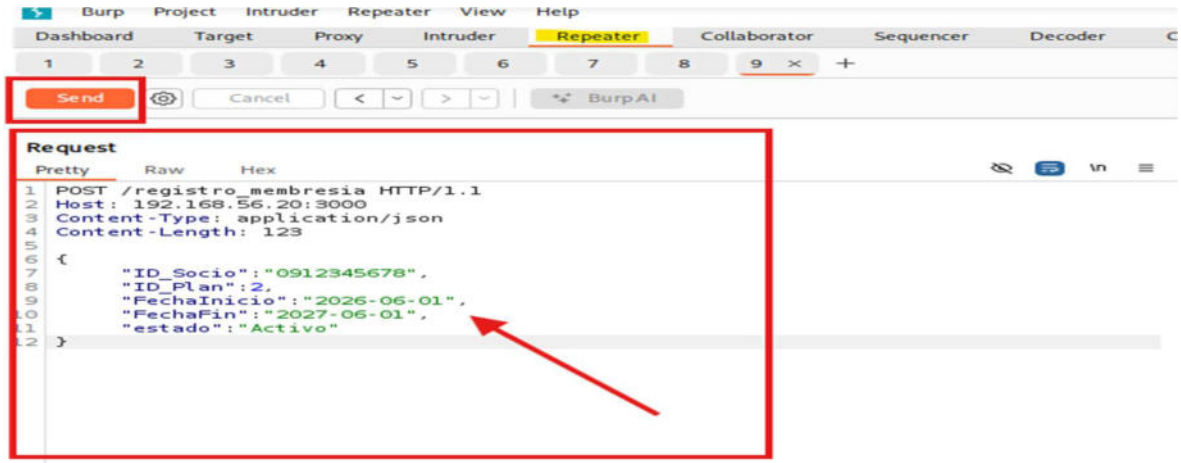
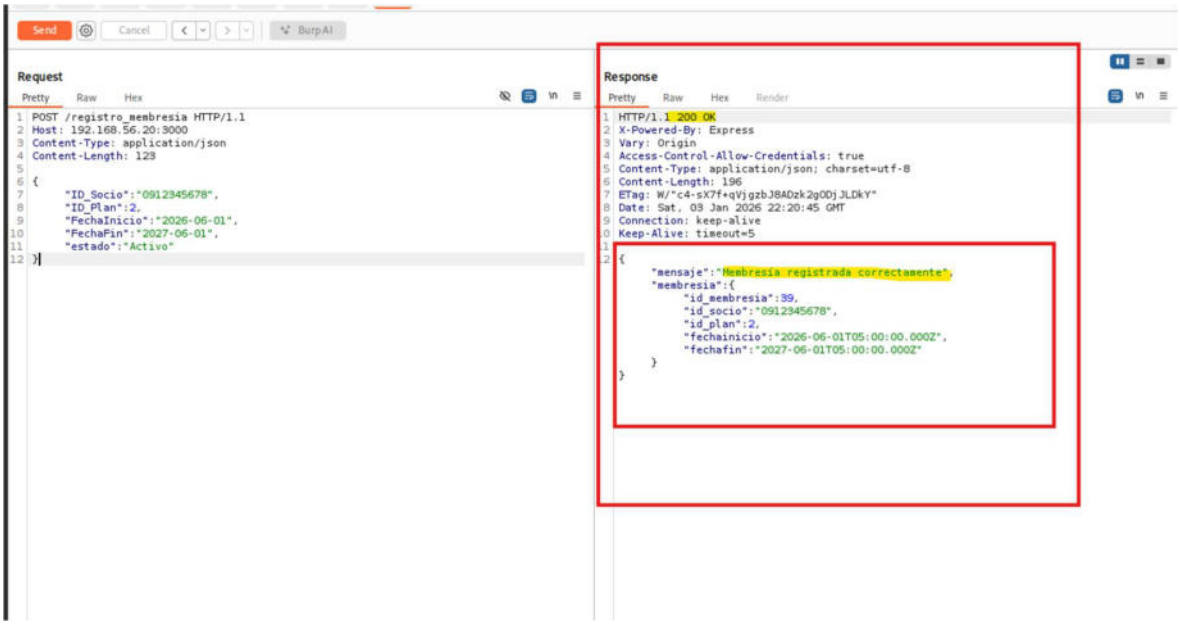


Figura 62:
Comprobación de registro.



Validamos que el código generado de forma manual funciona adecuadamente, procedemos en el mismo Brupsuite; a realizar el ataque desde el "Intruder" y cargamos el payload para la repetición del ataque.

Figura 63:
Ataque de Burpsuite con Intruder.

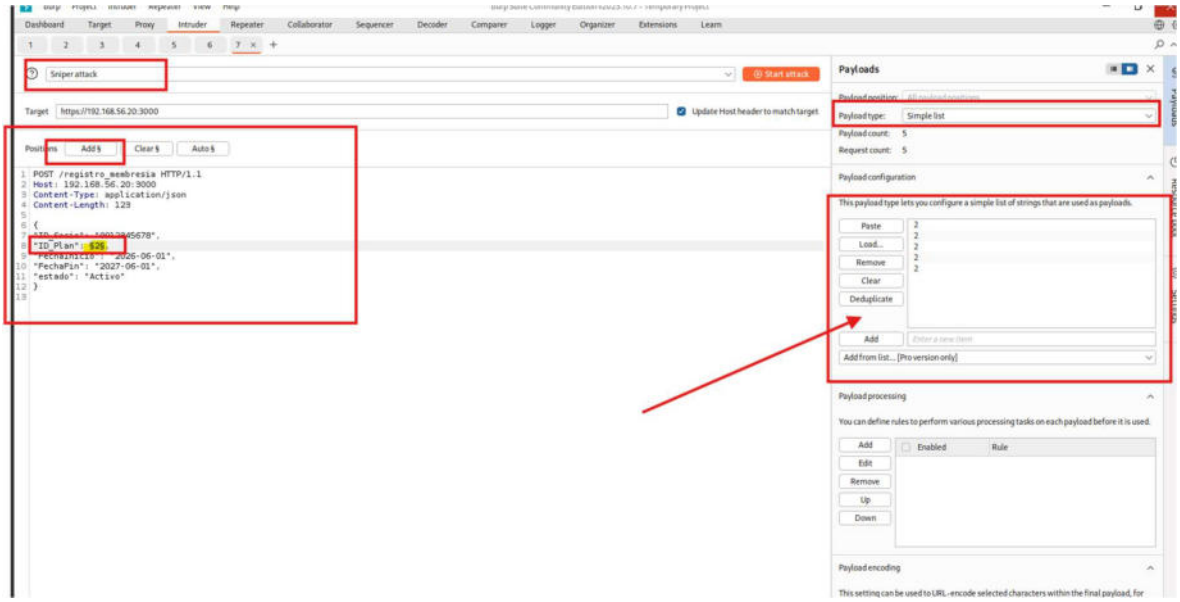


Figura 64:
Verificación de los nuevos registros desde Burpsuite con Intruder.

39	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
40	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
41	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
42	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
43	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
44	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z
45	0912345678	Juan	Pérez	Anual VIP	$360.00	2026-06-01T05:00:00.000Z	2027-06-01T05:00:00.000Z

Se encontraron 35 membresía(s).

7. Pruebas de Solicitudes del servidor (SSRF)

Referencia: API7:2023 - Server Side Request Forgery (SSRF)

7.1. Objetivo de la Prueba

Verificar que la API no acepta ni procesa URLs o rutas que apunten a recursos internos del servidor (ej. 127.0.0.1, archivos de sistema) o que obliguen al servidor a contactar con dominios de atacantes.

7.2. Escenario de Prueba

Identificar un endpoint que procese una URL o ruta proporcionada por el usuario (una función que descarga una imagen por URL, un webhook, o una función de importación).

7.3. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** Que al reemplazar la URL legítima con payloads internos y no registre interacciones.
- **Resultado Obtenido:** Ningún endpoint procesa URLs como esta en la documentación del Swagger.

8. Pruebas de Configuración Defectuosa

Referencia: API8:2023 - Security Misconfiguration

8.1. Objetivo de la Prueba

Identificar y documentar configuraciones inseguras a nivel de infraestructura, servidor web, framework o capa de aplicación que exponen información sensible o relajan los controles de seguridad predeterminados.

8.2. Escenario de Prueba

Identificar un endpoint que procese una URL o ruta proporcionada por el usuario (una función que descarga una imagen por URL, un webhook, o una función de importación).

8.3. Resultado Esperado vs. Obtenido

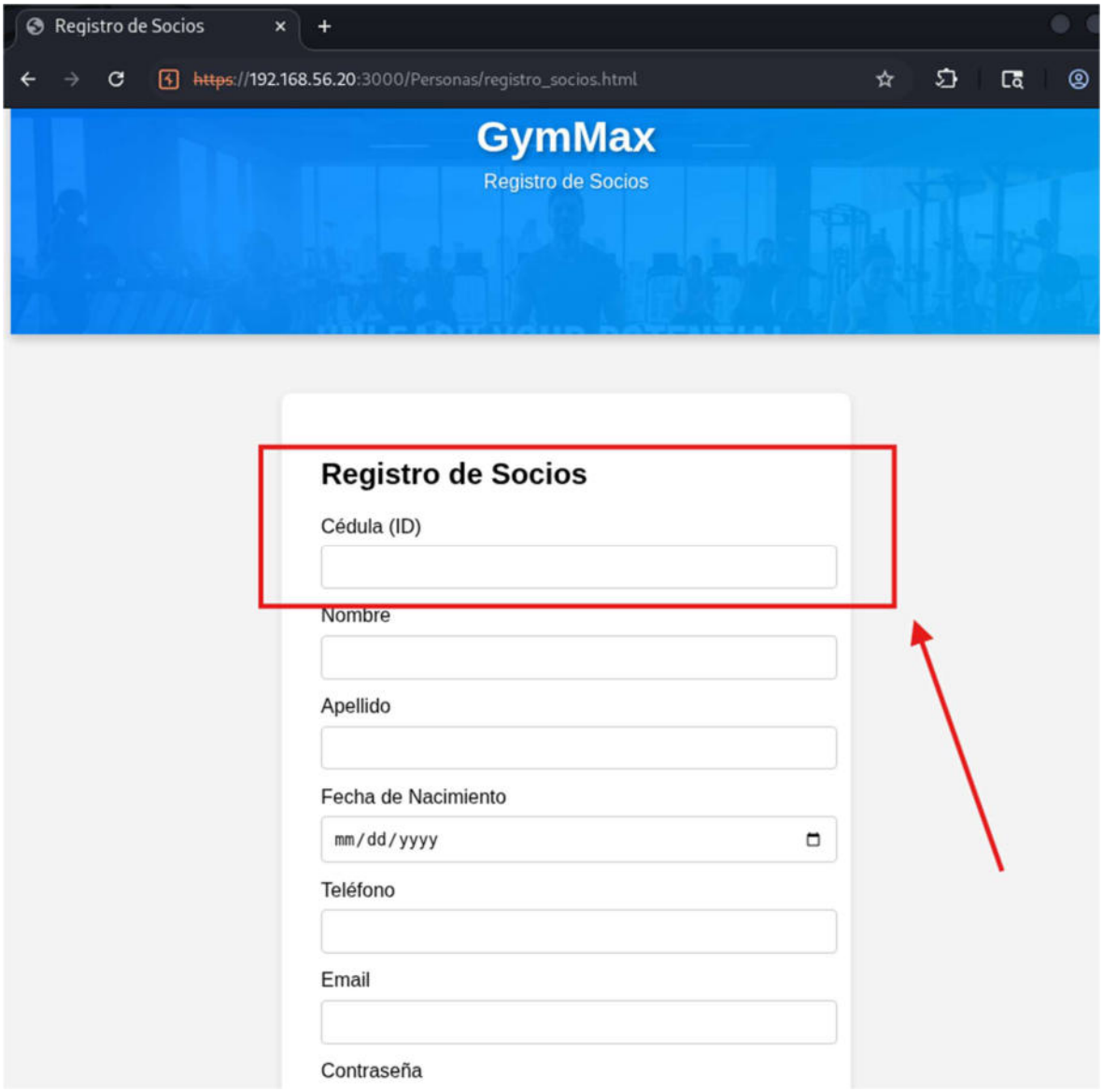
- **Resultado Esperado (Seguro):** Códigos de error apropiados (400, 404, 405) y el cuerpo solo contiene un mensaje genérico (ej. "Error inesperado" o "La ruta no existe"). Los mensajes de error no revelan información interna.
- **Resultado Obtenido:** En los distintos escenarios planteados se reflejan vulnerabilidades y en otros no, como vamos a detallar.

Para estas pruebas del API8, está dividido en varios pasos y/o escenarios; los cuales vamos a ir dettallando.

Escenario 1: Exposición de información por manejo de errores.

Fase 1: Vamos a ejecutar una petición POST que requiera validación.

Figura 65:
Registro de socios, ejecución de POST en los endpoints.



Procedemos al llenado de todos los campos, para posterior hacer la captura en Burpsuite antes del envió a guardar.

Figura 66:
Llenado de datos para el registro de socios.

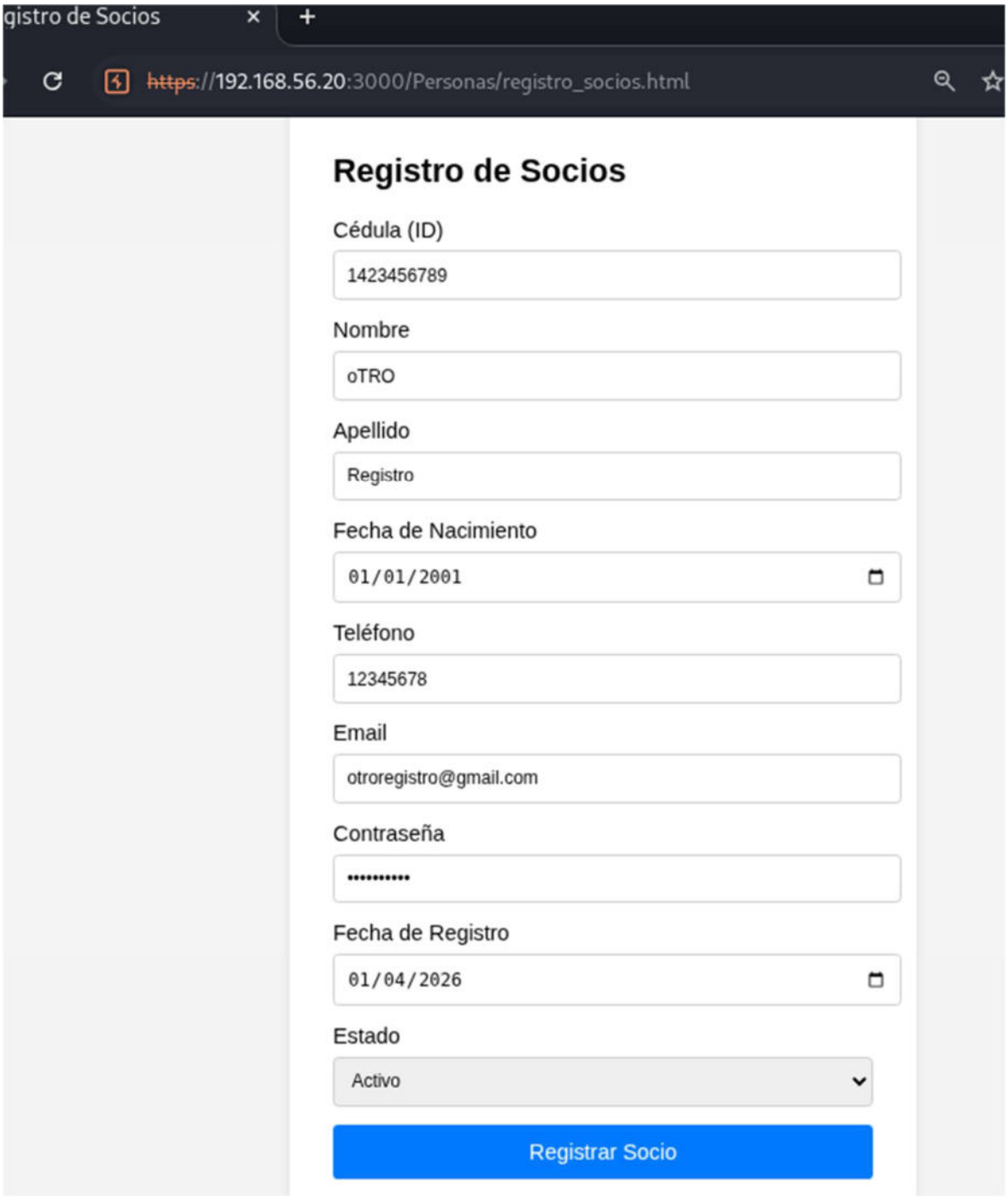


Figura 67:
Encendemos el “Intercept” en Burpsuite para realizar la captura del método POST.

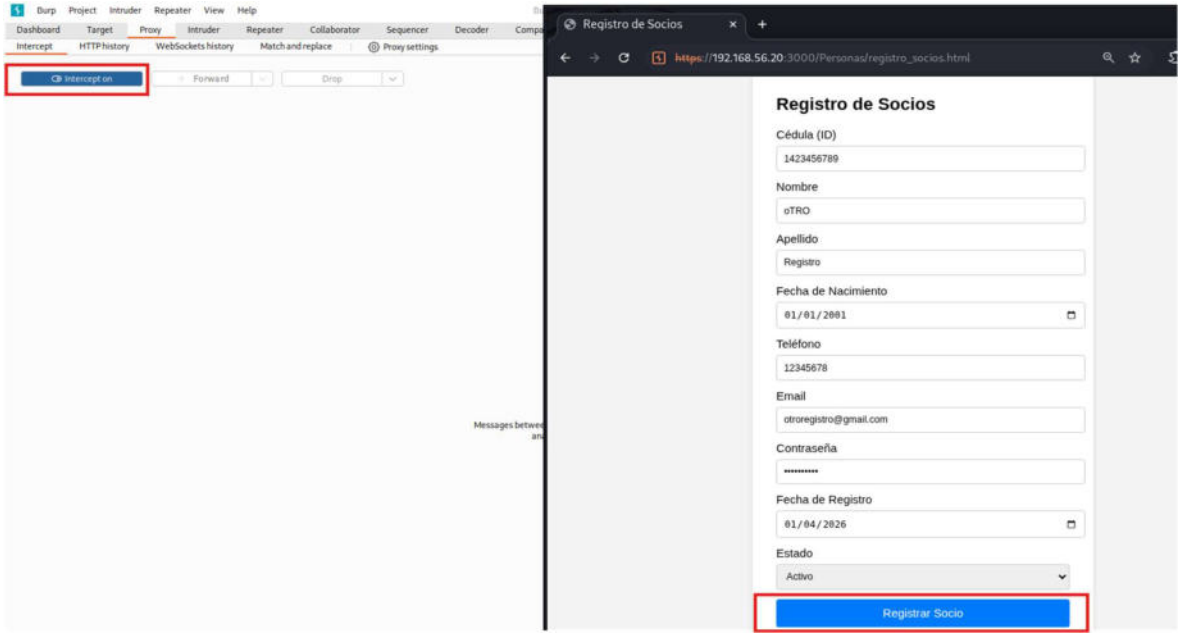
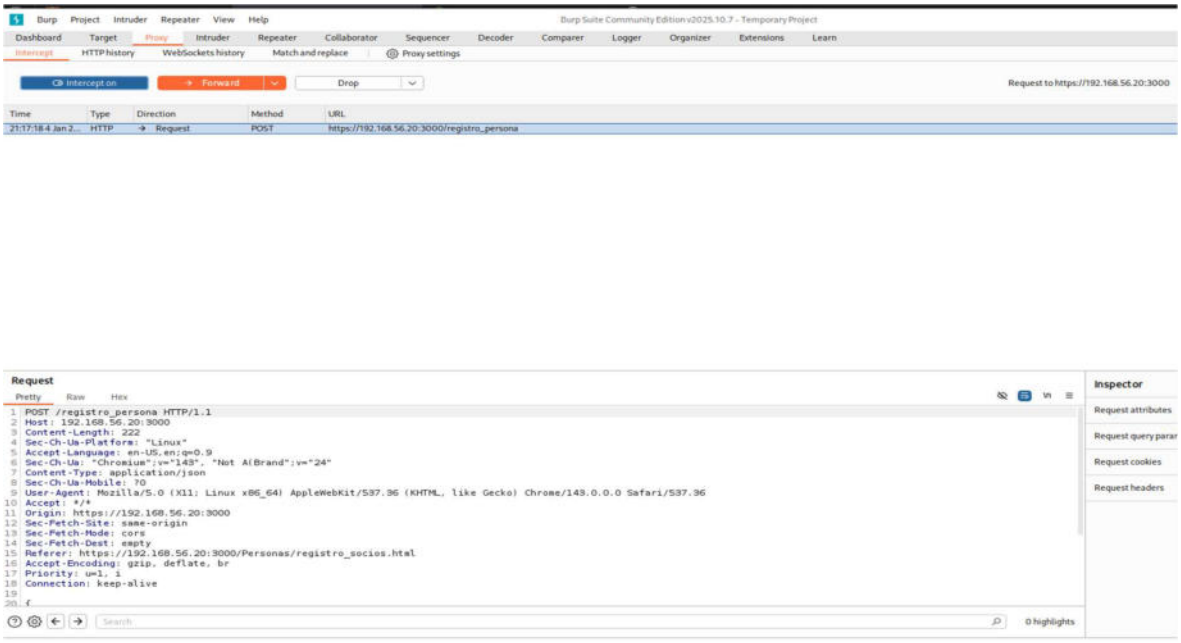


Figura 68:
Captura del método POST en Burpsuite.



Ahora procedemos al envío de lo interceptado a “Repeater”

Figura 69:
Envío lo capturado a “Repeater”.

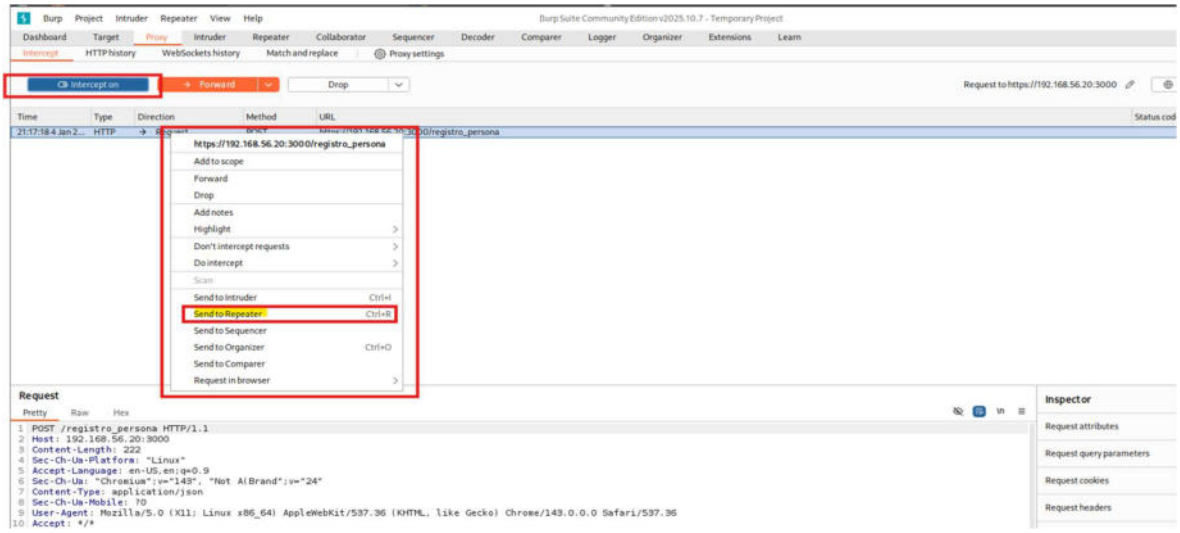
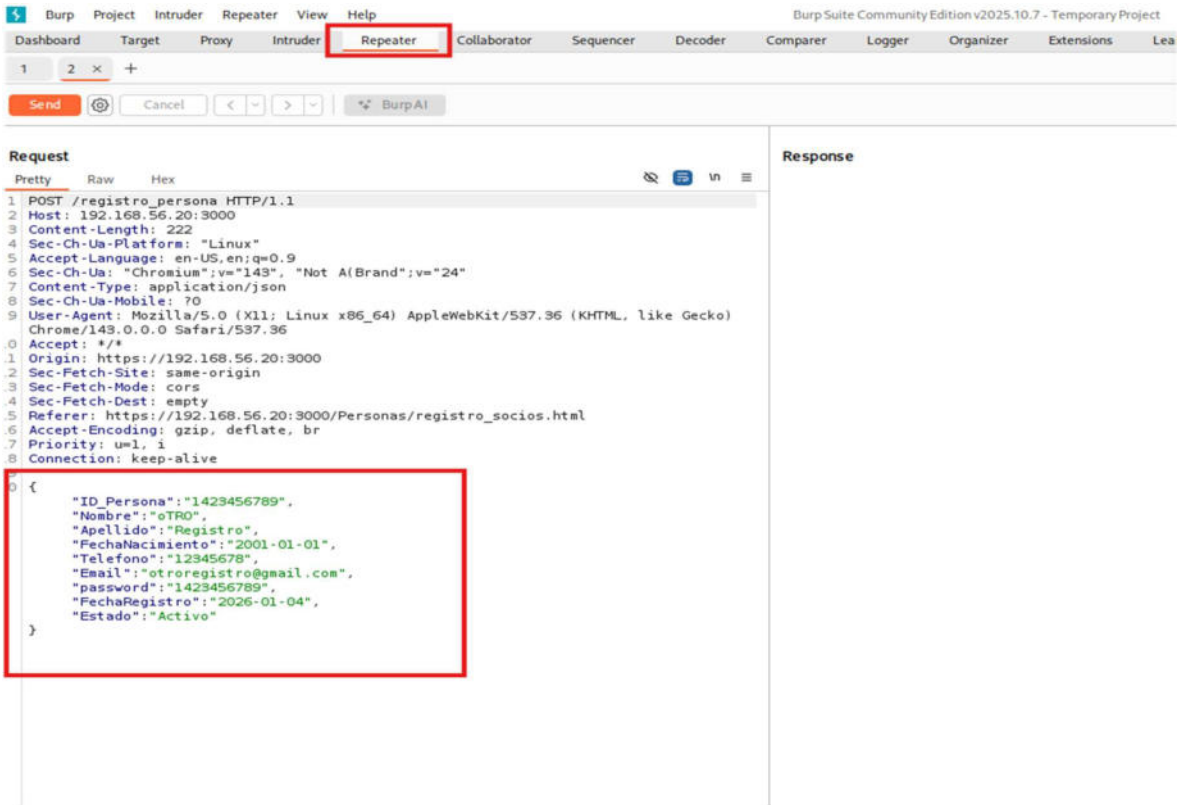
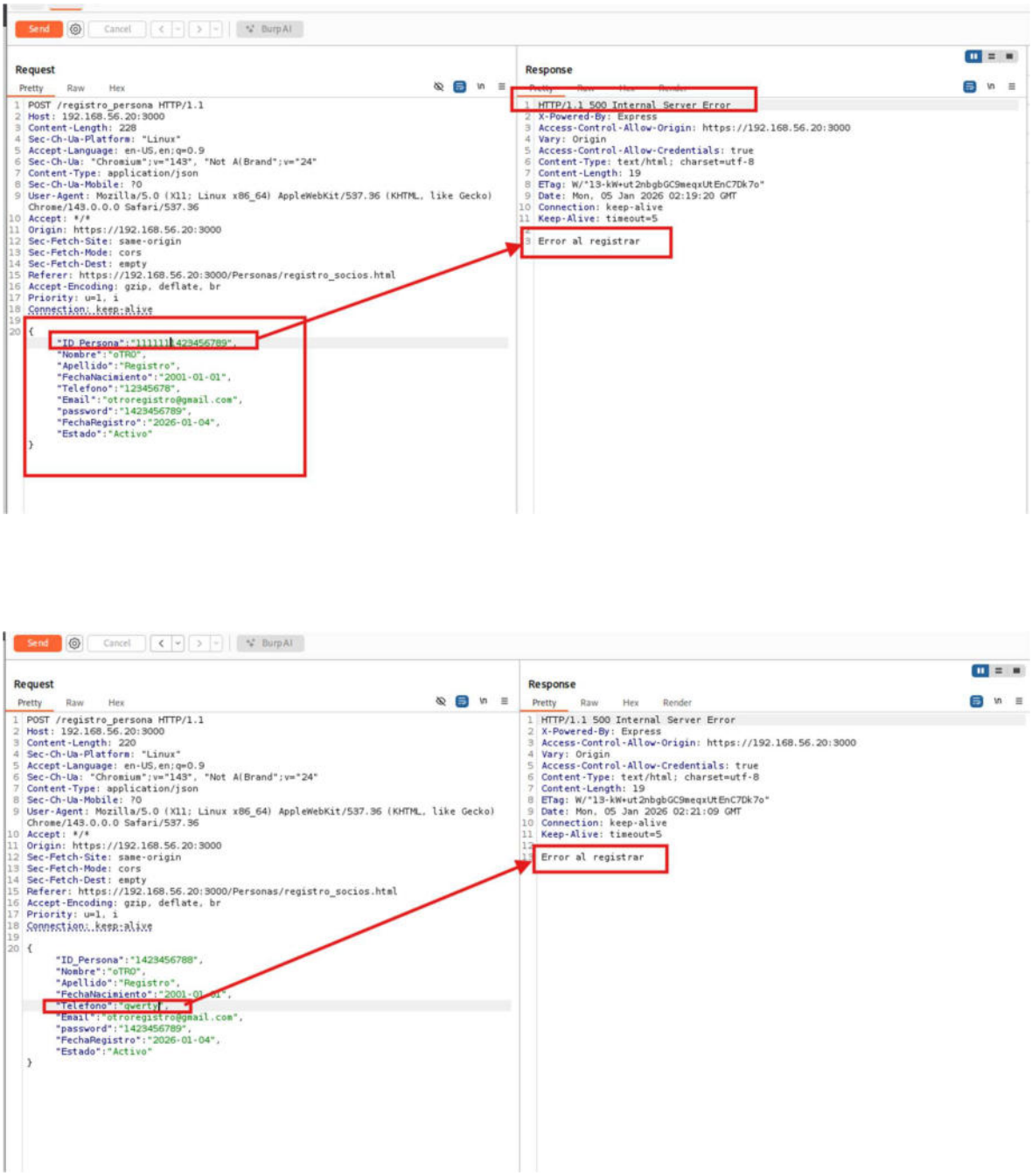


Figura 70:
Campos capturados y visualizados en “Repeater”.



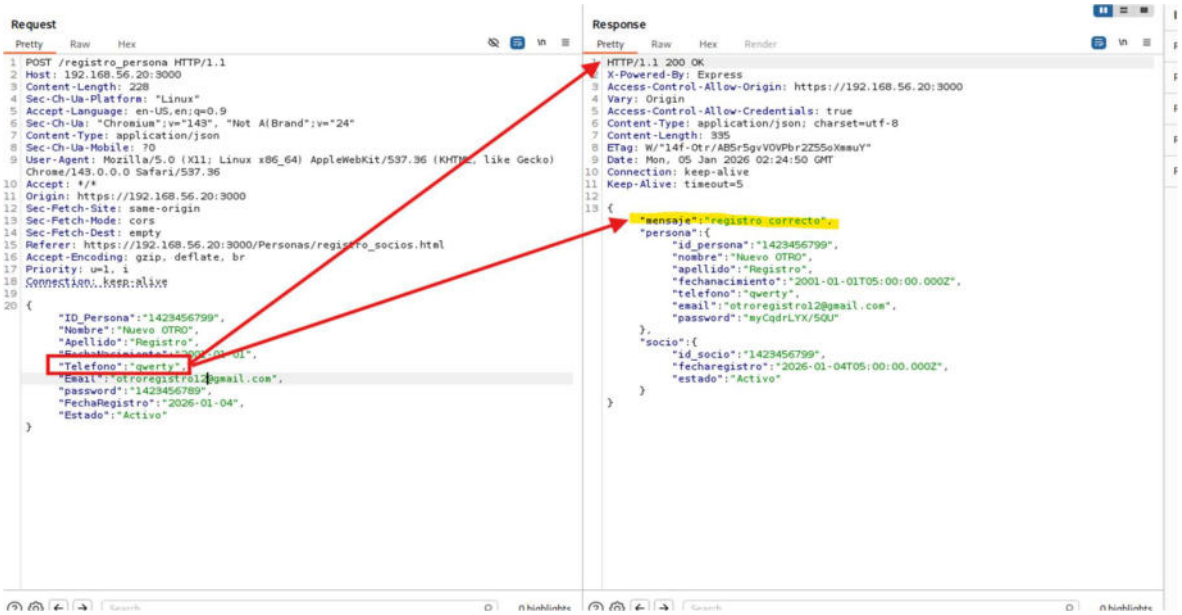
Capturado los datos, procedemos hacer pruebas cambiando y forzar el error, con la finalidad de validar que no permites los campos.

Figura 71:
Prueba cambiando los campos en “ID Persona” y “Telefono”.



Se repite la simulación y se realiza el envío de los datos y devuelve el código 200 OK, dando por registro correcto.

Figura 72:
Prueba cambiando los campos en “Telefono”.



Ahora procedemos a cambiar el estado en el registro y nos refleja el error, como se muestra en la figura 73.

Figura 73:
Prueba cambiando los campos en “Telefono”.

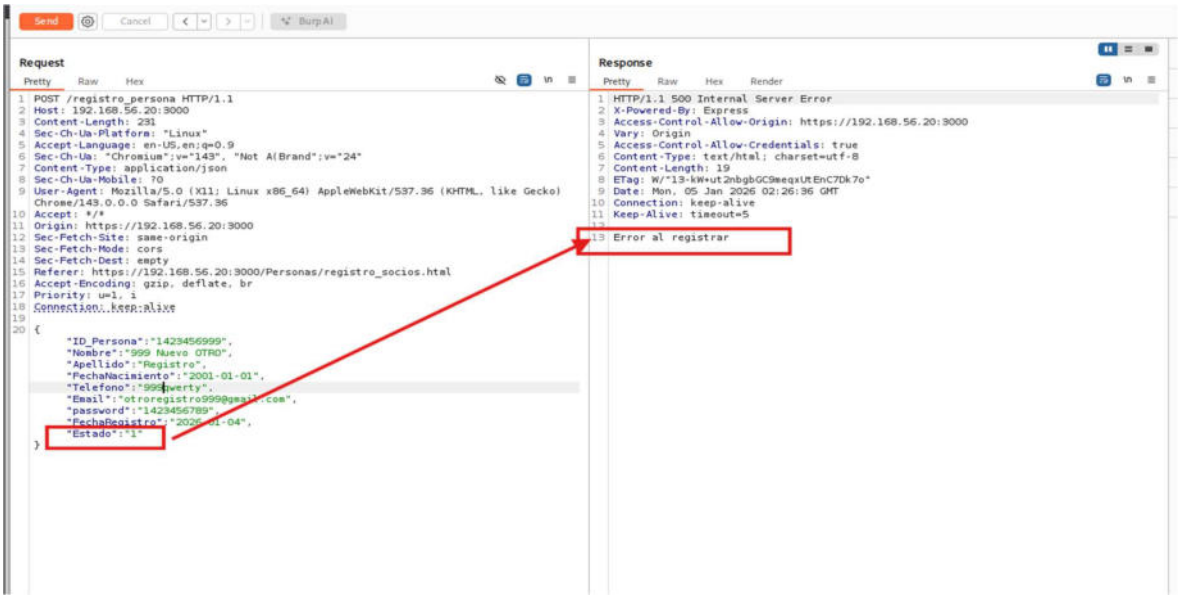
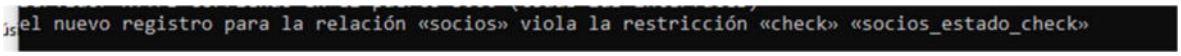


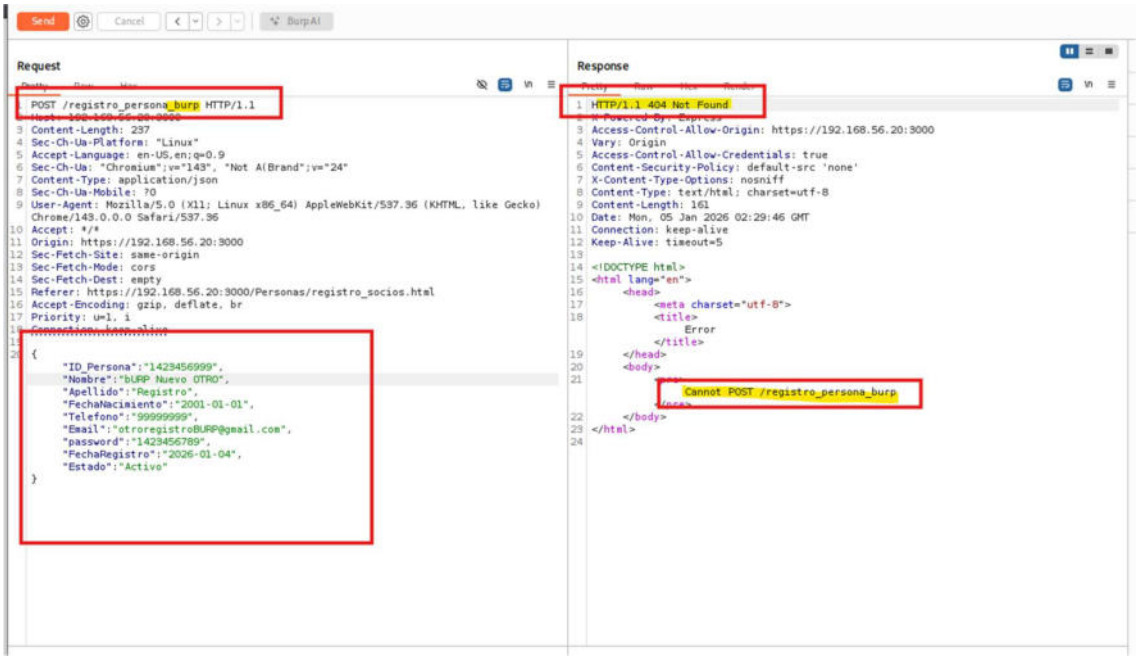
Figura 74:
Mensaje en el servidor de API al momento de cambiar el estado.



Fase 2: Forzar un error de ruta

Para esta fase, vamos a cambiar la ruta del endpoint por uno inexistente, tal como se muestra en la figura 75.

Figura 75:
Mensaje de ERROR ante una ruta inexistente.



Se considera que no tiene vulnerabilidades ya que los códigos de error apropiados (**400**, **404**, **405**) y el cuerpo solo contiene un mensaje genérico ("Error inesperado" o "La ruta no existe"). Los mensajes de error no revelan información interna, es su principal factor para que no se vulnerable.

Escenario 2: Análisis de Headers de Respuesta y Versiones.

Fase/Paso 1: Recolectar Headers.

Para la recolección de headers, vamos a procer a realizar y ejecutar los endponit, para la captura en Burpsuite como se detalla a continuación.

Figura 76:
Ingreso al menú maquinas.

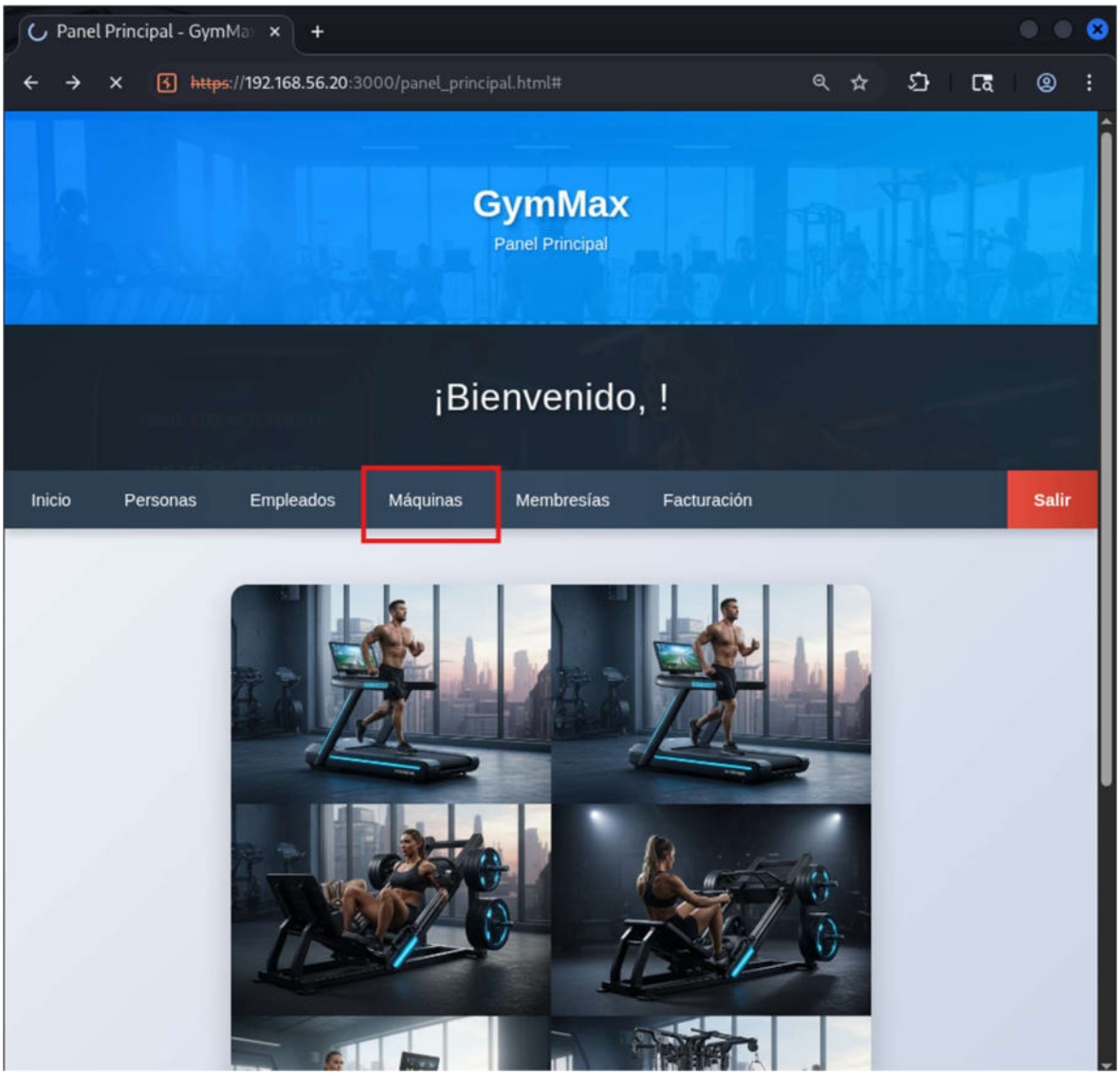
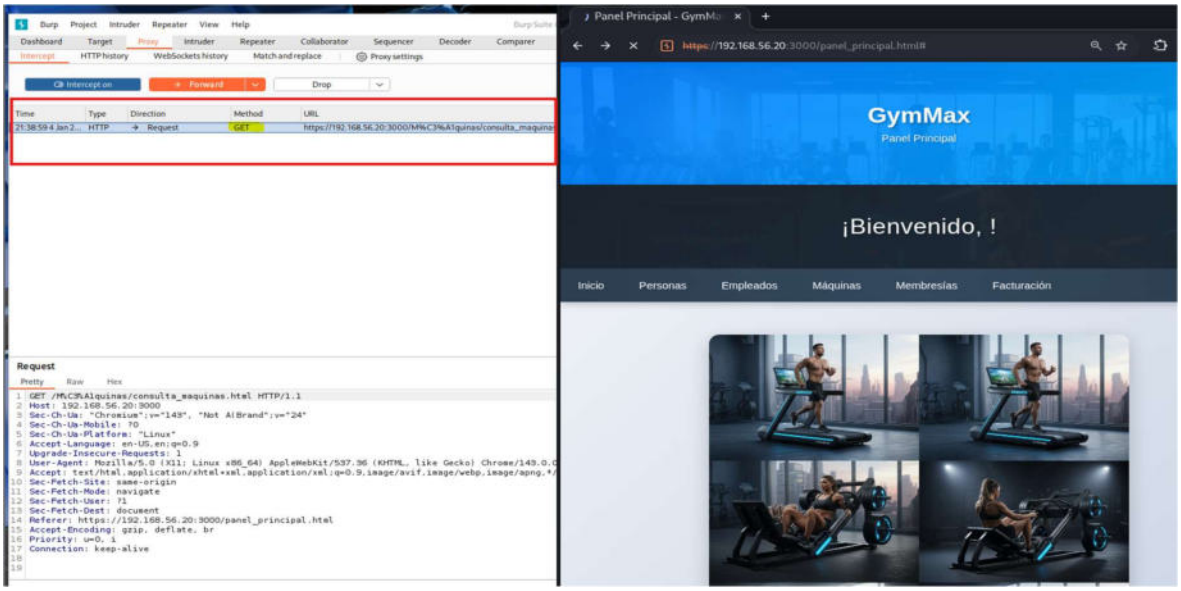


Figura 77:
Captura en Burpsuite el método GET.



Posterior a la captura en Burpsuite procedemos el envío a “Repaeter”, como se muestra en la figura 78.

Figura 78:
Envío a “Repeater” el método GET.

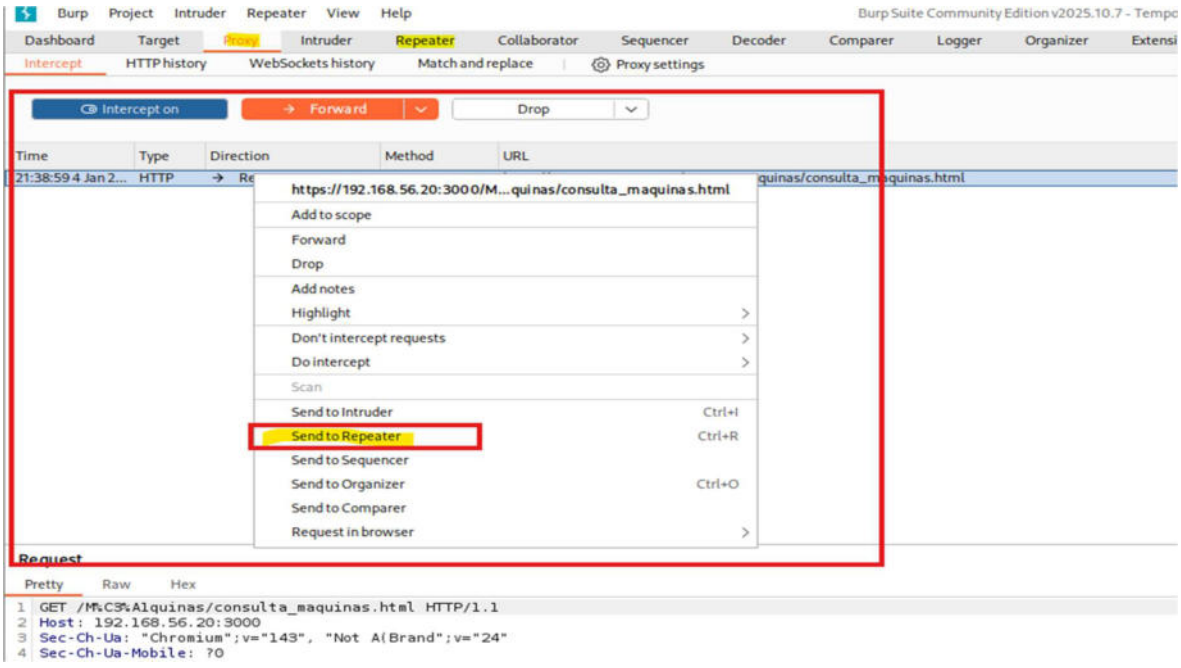
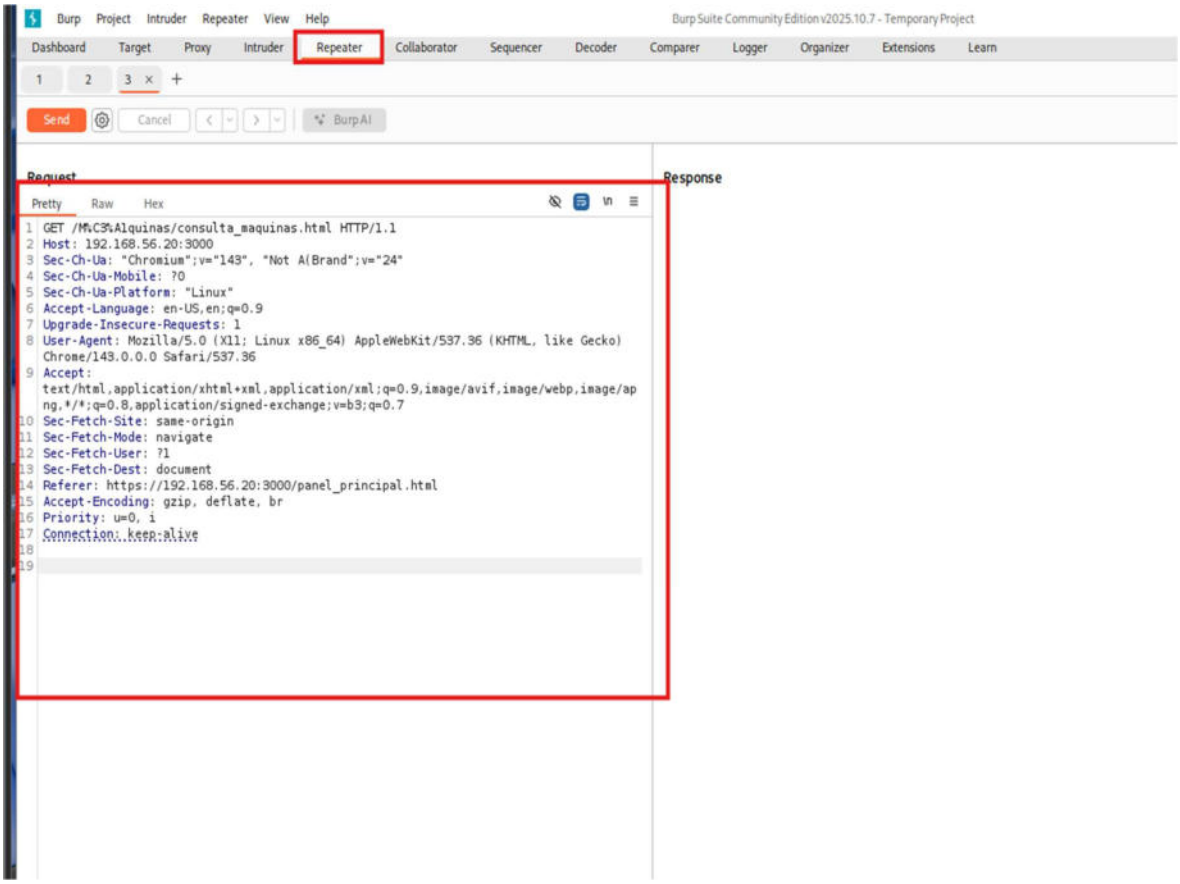


Figura 79:
Análisis en “Repeater” el método GET en Burpsuite.



Durante la prueba que hice se esta recolectando cabeceras del HTTP únicamente, asociadas a la solicitud propia; las cuales permite analizar configuraciones.

Fase/Paso 2: Revisar la exposición de versiones.

Para la recolección de headers, vamos a realizar y ejecutar los endpoint, para la captura en Burpsuite como se detalla a continuación.

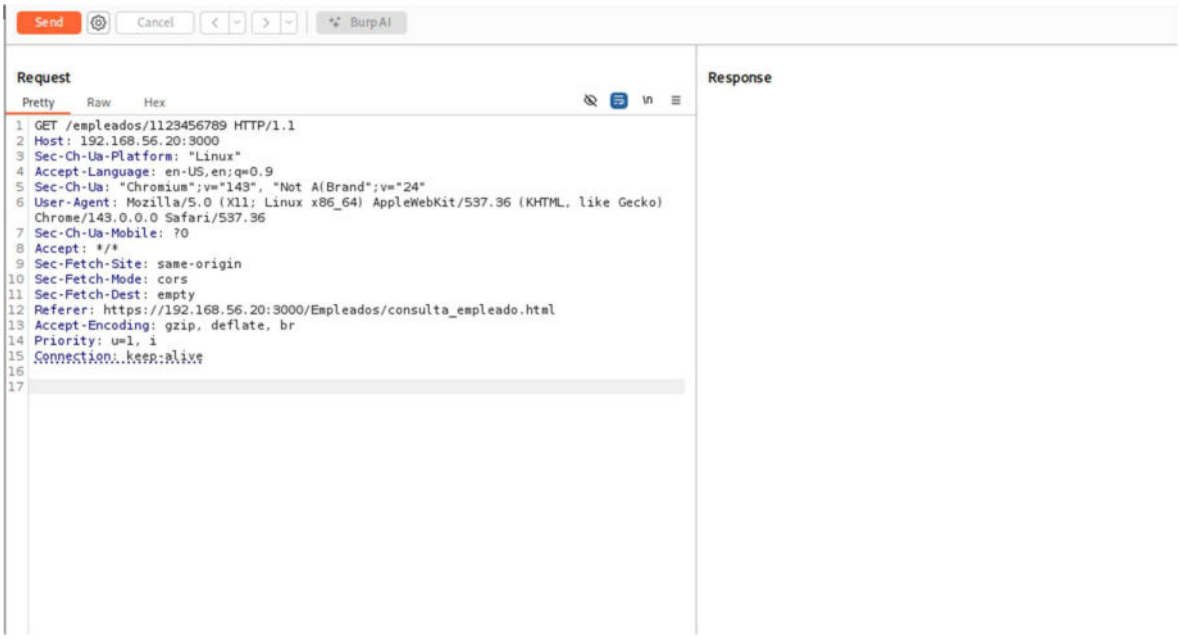
Sin embargo, para este escenario y fase/paso 2; no es vulnerable debido a que no se ha realizado versionamiento por lo cual no va a revelar el mismo.

Fase/Paso 3: Revisar headers de seguridad críticos.

Para la recolección de headers, vamos a proceder a realizar y ejecutar los endpoint, para la captura en Burpsuite como se detalla a continuación.

Continuaremos analizando lo capturado en Burpsuite, como se muestra en la figura 80.

Figura 80:
Análisis en "Repeater" el método capturado en Burpsuite.



En base al análisis de la captura en Burpsuite, determinamos que es vulnerable en base a la siguiente tabla.

Tabla 22:
Resumen de Riesgo

Header	Estado en la imagen	Riesgo Identificado
Strict-Transport-Security (HSTS)	**Ausente**	No se detalla una instrucción para forzar HTTPS. Además, la petición se dirige a una dirección IP directa (192.168.56.20 servidor), lo cual suele implicar el uso de HTTP simple (puerto 3000 sin TLS), permitiendo ataques de downgrade.

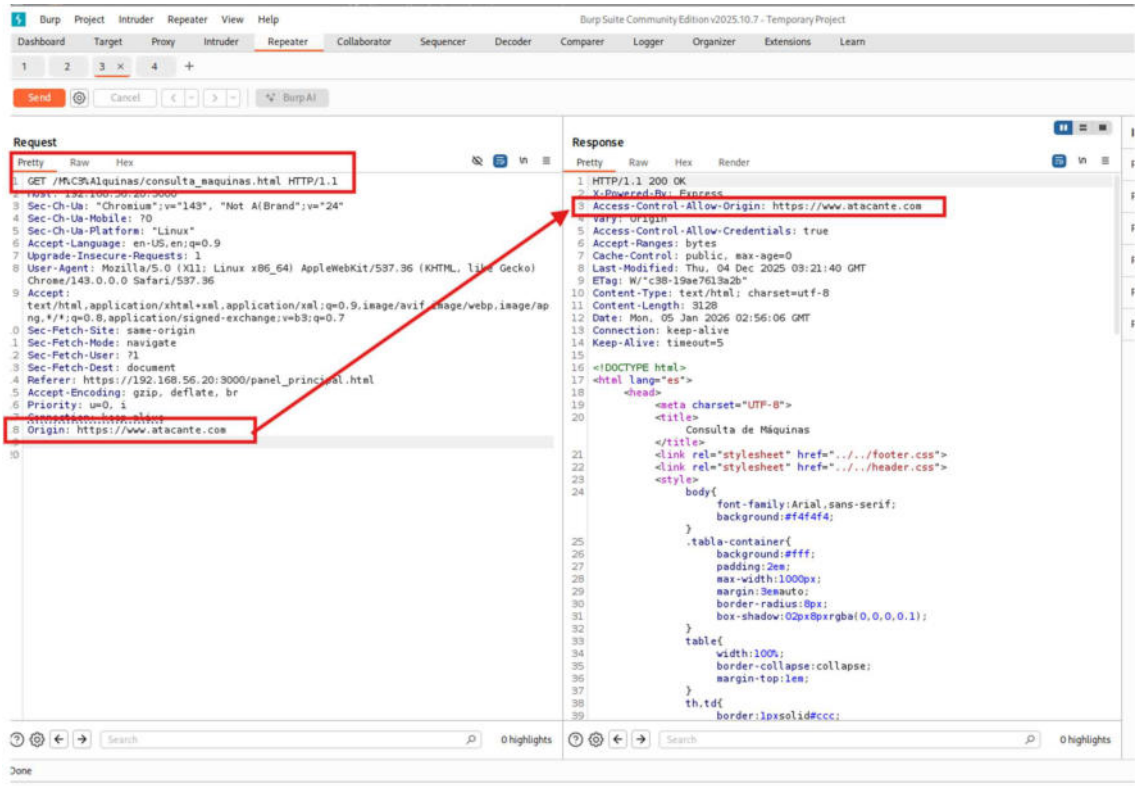
Header	Estado en la imagen	Riesgo Identificado
Content-Security-Policy (CSP)	**Ausente**	El cliente realiza una petición cors, pero no hay políticas de seguridad de contenido visibles que restrinjan de dónde se pueden cargar scripts o recursos, aumentando el riesgo de XSS.
X-Frame-Options	**Ausente**	Al no estar definido en la configuración de la sesión visible, el sitio no tendría protección contra ataques de clickjacking (no se prohíbe el renderizado en iframes).

Escenario 3: Configuración insegura de CORS y métodos HTTP.

Fase/Paso 1: Prueba de CORS.

Procedemos a realizar la captura en Burpsuite del método GET y lo enviamos a “Repeater”, para posterior ingresar un dominio malicioso (simulación); como se muestra en la figura

Figura 81:
Análisis en “Repeater” el método capturado en Burpsuite.



Cuando realizamos el cambio, nos damos cuenta que al enviar desde Burpsuite el dominio devuelve valores el cual hace determinar la vulnerabilidad de esta prueba.

Fase/Paso 2: Prueba de métodos HTTP no autorizados.

Para estas pruebas procedemos a capturar el método GET y lo pasamos a REPEATER, para lo cual vamos a cambien el método de GET a OPTIONS, como se refleja en las figuras siguientes.

Figura 82:
Captura de Burpsuite y enviado a “Repeater”.

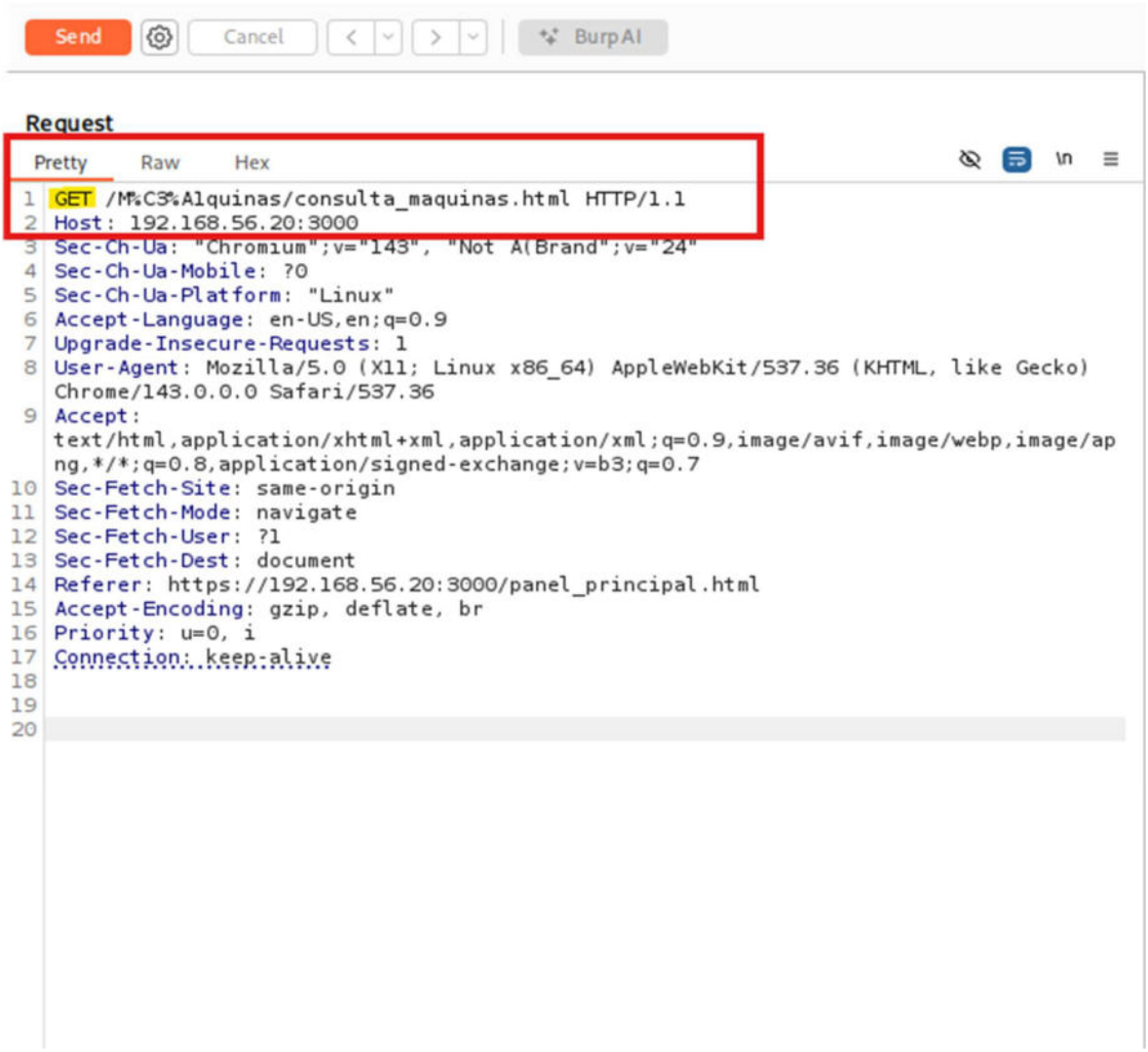
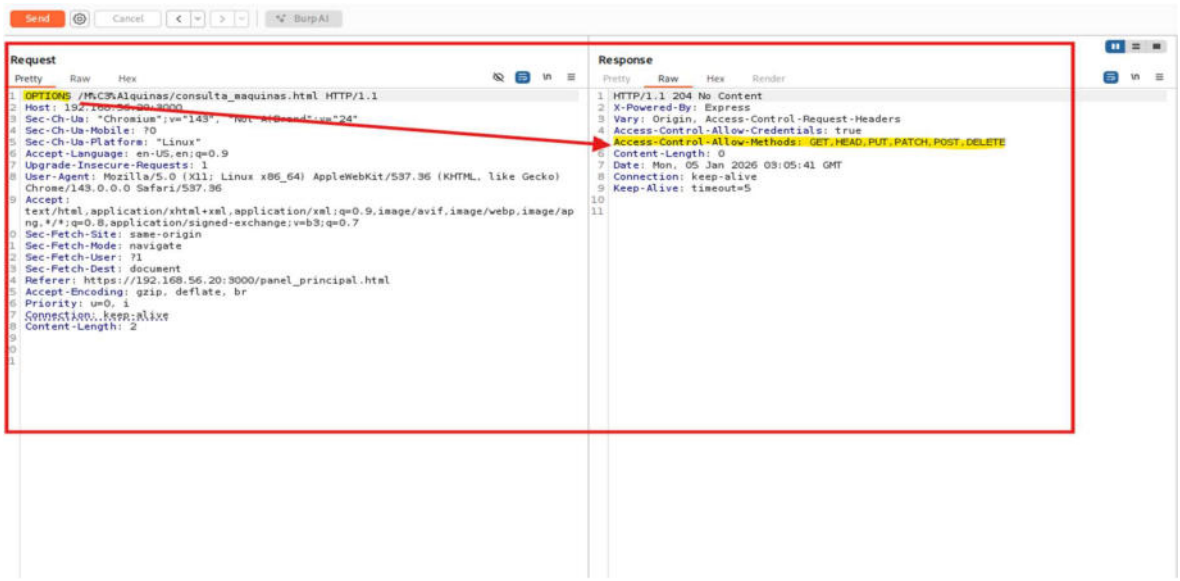


Figura 83:
Cambio del método en Burpsuite de GET a OPTIONS.



Para determinar si es vulnerable o no, procedemos a cambiar ahora el método por TRACE, como se muestra en la figura 84.

Figura 84:
Cambio del método en Burpsuite por TRACE.

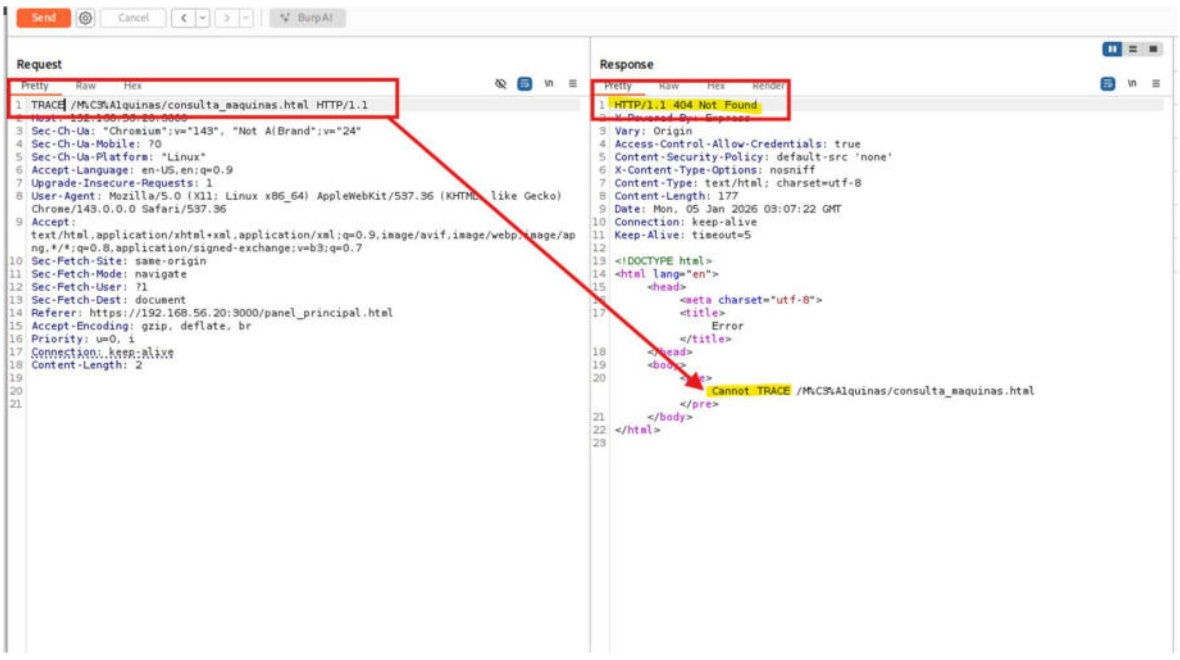


Figura 85:
Cambio del método en Burpsuite por PUT.

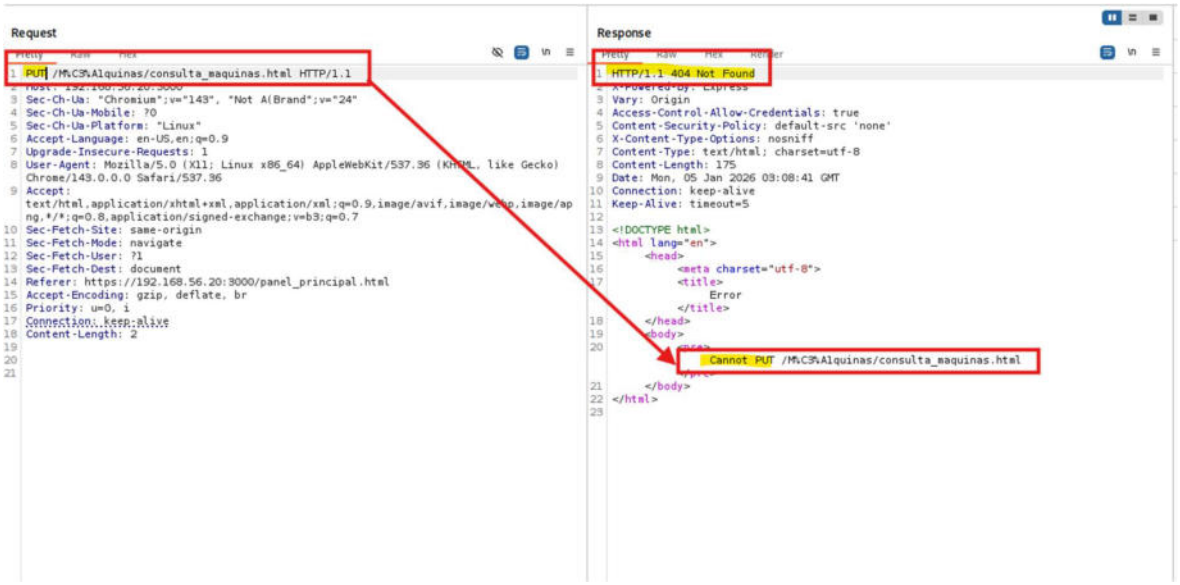
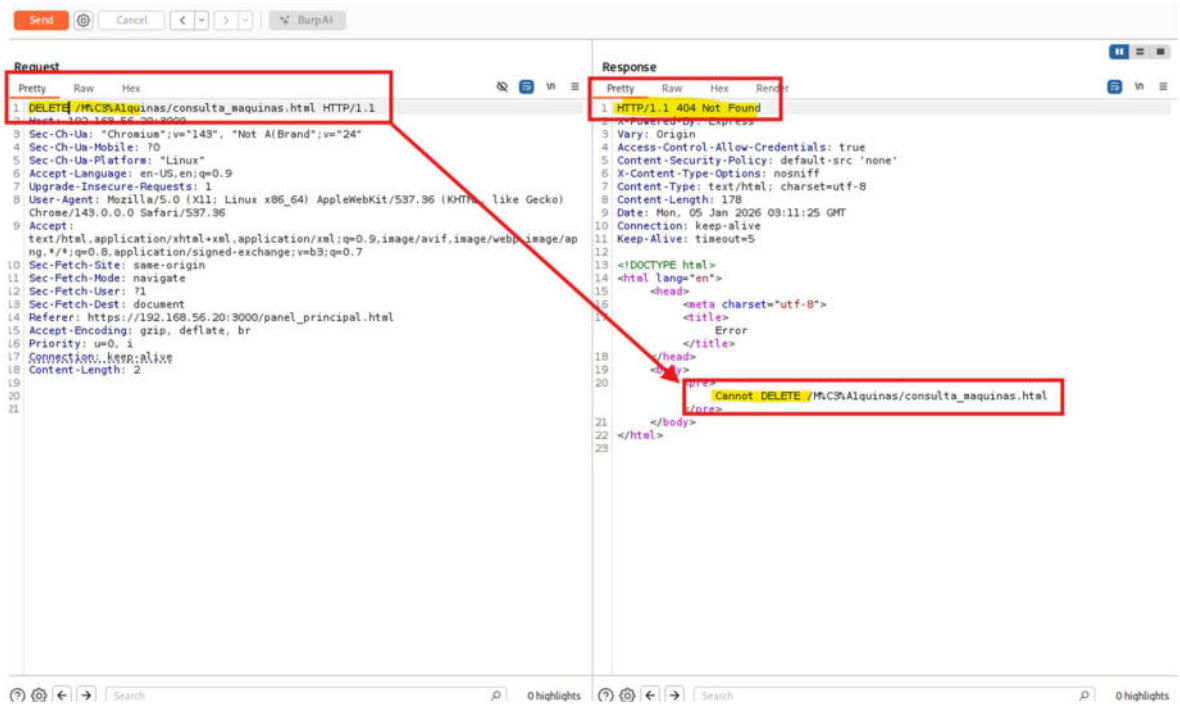


Figura 86:
Cambio del método en Burpsuite por DELETE.



Una vez realizado todos los cambios, para todos los casos la respuesta es de 404, lo cual concluimos que para este escenario no es vulnerable.

9. Pruebas de Gestión de Inventario

Referencia: API9:2023 - Improper Inventory Management (Gestión de Inventario Defectuosa)

9.1. Objetivo de la Prueba

Identificar y explotar endpoints obsoletos, versiones API sin mantenimiento, o funciones internas no documentadas.

9.2. Escenario de Prueba

Identificar y envíar una solicitud a un endpoint versionado.

9.3. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** Que no se enliste las versiones.
- **Resultado Obtenido:** No aplica, ya que solo se tiene la primera versión.

Para estas pruebas del API9, no aplica, ya que corresponde a nuestra primera versión.

10. Pruebas de Consumo Inseguro de API's

Referencia: API10:2023 - Unsafe Consumption of API's

10.1. Objetivo de la Prueba

Identificar una solicitud de consumo de datos de otras API's.

10.2. Escenario de Prueba

Identifica una solicitud donde se sospeche que los datos se reenvían a un servicio de terceros.

10.3. Resultado Esperado vs. Obtenido

- **Resultado Esperado (Seguro):** La API valide y filtre toda la entrada antes de confiar en la misma.
- **Resultado Obtenido:** No aplica, ya que no se consume datos de terceros.

Para estas pruebas del API10, no aplica, ya que no se consume datos de terceros.

CAPITULO 6

6. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

- El presente trabajo nos permitió adentrarnos en el mundo de la seguridad de las APIs, lo que actualmente es fundamental considerando su uso a gran escala en todas las organizaciones y por otro lado el usar OWASP API Top 10 y NIST como marcos de referencia nos permitió un mejor enfoque, así como una cobertura exhaustiva de los riesgos más críticos específicos de las arquitecturas modernas de las APIs.

- Con la metodología propuesta y utilizada, la API fue probada donde residen sus debilidades más comunes: el control de acceso a datos y funciones, además nos permitió buscar fallos en su arquitectura, evaluando además si existen versiones obsoletas o *endpoints* de prueba que han quedado expuestos.

- El uso de Burp Suite y OWASP ZAP en la evaluación de una API RESTful permitió complementar la evaluación alcanzando una mejor cobertura técnica. Con Owasp Zap pudimos descubrir los *endpoints,* así como detectar de manera automatizada fallos de configuración (API8: Misconfiguration) y vulnerabilidades de inyección de bajo riesgo, permitiendo establecer una línea base de seguridad rápidamente. Por otro lado, Burp Suite (específicamente Repeater e Intruder) es muy útil para detectar vulnerabilidades de lógica de negocio (API1: BOLA, API5: BFLA). La posibilidad de manipular manualmente *tokens*, *cookies* y parámetros *ad-hoc* en Repeater es útil para probar las fallas de autorización que el escaneo automatizado no puede detectar.

- La importancia y los beneficios de definir un marco estructurado utilizando como referencia OWASP API Top 10 y NIST radican en transformar un proceso de prueba reactivo y ad-hoc en una estrategia de gestión de riesgos proactiva, medible y

auditable. Usar un enfoque sistemático y estructurado garantiza que las pruebas no se limiten a fallos de inyección comunes (XSS, SQLi), sino que se concentren en los riesgos más significativos de las APIs, como los fallos de autorización (BOLA y BFLA), que son la causa principal de las filtraciones masivas de datos, además permite replicar la calidad de la prueba en cada ciclo de vida de la API (Dev, QA, Pre-Prod). Esto asegura que los endpoints no se desplieguen con fallos conocidos de versiones anteriores y obliga a la verificación sistemática de controles de seguridad que a menudo se pasan por alto como la configuración de headers, manejo de errores, Rate Limiting), previniendo los riesgos de una configuración defectuosa.

- La implementación estructurada de las evaluaciones de seguridad en etapas iniciales del desarrollo (DevSecOps) representa el mayor beneficio a largo plazo, ya que identificar y corregir las vulnerabilidades en las etapas de diseño o prueba es mucho más económico que hacerlo después de la salida a producción, evitando correcciones de emergencia así como posibles afectaciones al impacto reputacional y hace que la seguridad sea una cualidad inherente en las APIs, blindando la lógica de negocio y protegiendo los datos críticos frente a ataques.

6.2. Recomendaciones de Mitigación

Con la intención de mitigar las vulnerabilidades identificadas con las recomendaciones de seguridad propuestas se acompañan de un plan de implementación y métricas para validar su efectividad:

1. Fortalecimiento de autenticación y autorización:

Implementar OAuth 2.0/OpenID Connect con rotación periódica de tokens.

Métrica: reducción del 100% de accesos no autenticados identificados en pruebas posteriores.

2. Protección de datos sensibles:

Aplicar cifrado en tránsito (TLS 1.3) y en reposo (AES-256).

Métrica: validación mediante escaneo automatizado de que no existen endpoints transmitiendo datos sin cifrar.

3. Rate limiting y protección contra abusos:

Configurar políticas de rate limiting (ej. 100 requests/min por usuario).

Métrica: pruebas de carga deben demostrar que el sistema rechaza excesos sin degradar disponibilidad global.

4. Gestión de errores y registros:

Implementar mensajes de error genéricos y centralizar logs en SIEM para monitoreo proactivo.

Métrica: revisión de logs asegura ausencia de exposición de stack traces sensibles.

6.3. Recomendaciones para mitigar riesgos en APIs relacionadas con OWASP

API1:2023 Autorización de nivel de objeto roto

- Implementar un sistema de autorización que respete las políticas internas y la jerarquía de roles del usuario.
- Verificar en cada función que utilice datos proporcionados por el cliente que el usuario autenticado realmente tiene permisos para realizar la acción sobre el recurso solicitado.

- Utilizar identificadores aleatorios y no predecibles, como GUID, para los registros con el fin de reducir el riesgo de manipulación de IDs.
- Diseñar y mantener pruebas de seguridad orientadas a validar el correcto funcionamiento del mecanismo de autorización, evitando aplicar cambios que provoquen fallos en estas pruebas.

API2:2023 Autenticación rota

- Identificar todos los flujos de autenticación de la API (móvil, web, enlaces profundos, inicio de sesión con un clic, etc.) y verificar con el equipo técnico que no falte ninguno.
- Comprender claramente los mecanismos de autenticación empleados y su función real; tecnologías como OAuth o las API keys no sustituyen un proceso de autenticación completo.
- Adoptar soluciones estándar para autenticación, emisión de tokens y gestión de contraseñas, evitando desarrollos propios innecesarios.
- Proteger los endpoints de recuperación de cuentas con los mismos controles que un inicio de sesión: limitación de intentos, bloqueo temporal y medidas contra fuerza bruta.
- Solicitar autenticación adicional para acciones altamente sensibles, como cambiar correo, contraseña o datos asociados a 2FA.
- Aplicar defensas contra fuerza bruta y robo de credenciales, empleando límites estrictos, bloqueo de cuentas, captchas y controles de contraseñas débiles.

API3:2023 Autorización de nivel de propiedad de objeto roto

- Controlar el acceso a propiedades: Asegurarse de que el usuario solo pueda ver o modificar los atributos del objeto a los que tiene permiso.
- Evitar métodos genéricos: No usar funciones como to_json() o to_string() que expongan todas las propiedades; seleccionar únicamente los campos necesarios.
- Prevenir asignación masiva: No vincular automáticamente la entrada del cliente a variables internas o propiedades sensibles.
- Validación y minimización de datos: Limitar los datos devueltos al mínimo requerido y utilizar esquemas de validación para garantizar que la respuesta cumpla con las reglas de seguridad.

API4:2023 Consumo de recursos sin restricciones

- Control de recursos del sistema: Usar entornos como contenedores o funciones sin servidor para limitar de manera automática memoria, CPU y otros recursos críticos.
- Restringir el tamaño de entrada: Definir límites claros para todos los datos recibidos (longitud de cadenas, número de elementos, tamaño de archivos).
- Aplicar limitación de solicitudes: Establecer reglas de rate limiting ajustadas a las necesidades del negocio, especialmente para operaciones sensibles o fácilmente abusables.
- Regulares operaciones repetitivas: Restringir la frecuencia con la que un usuario puede ejecutar acciones específicas, como validar un OTP o solicitar recuperación de contraseña.
- Validación en el servidor: Verificar parámetros que influyen en la cantidad de datos devueltos, evitando que el cliente solicite volúmenes excesivos de información.

API5:2023 Autorización de nivel de función rota

- Centralizar la autorización: La aplicación debe contar con un módulo de control de acceso uniforme, fácil de revisar y aplicado en todas las funciones del negocio, ya sea integrado o provisto por componentes externos.
- Acceso denegado por defecto: Todo endpoint debe requerir permisos explícitos; ningún usuario o rol debería tener acceso implícito.
- Revisión de endpoints: Evaluar sistemáticamente los puntos finales para detectar fallas de autorización a nivel de función, considerando la lógica de negocio y la jerarquía de roles o grupos.
- Controles en funciones administrativas: Asegurar que los controladores y métodos administrativos implementen validaciones basadas en roles, ya sea heredando de controladores especiales o aplicando comprobaciones directas.

API6:2023 Acceso sin restricciones a flujos comerciales sensibles

La planificación de la mitigación debe realizarse en dos capas:

- Negocios: identificar los flujos de negocios que podrían dañar al negocio si se utilizan en exceso.
- Ingeniería: elegir los mecanismos de protección adecuados para mitigar el riesgo empresarial.

Para mitigar ataques automatizados, se emplean diversos mecanismos que dificultan o encarecen la actividad de los bots. Algunos son simples de aplicar y otros requieren soluciones más avanzadas, pero todos buscan identificar comportamientos anómalos o no humanos.

Entre las técnicas más utilizadas destacan:

- Huella del dispositivo: Identificar y bloquear clientes sospechosos (navegadores sin interfaz gráfica) para obligar al atacante a usar herramientas más complejas.
- Verificación humana: Incorporar captchas o métodos biométricos que diferencien a un usuario real de un bot.
- Detección de patrones automatizados: Analizar la secuencia y velocidad de las acciones del usuario para descubrir comportamientos imposibles para una persona (por ejemplo, completar un proceso de compra en fracciones de segundo).
- Restricciones por red: Filtrar o bloquear IP asociadas a Tor o a proxys públicos conocidos.
- Protección de APIs orientadas a máquinas: Asegurar y limitar el acceso a APIs usadas por desarrolladores o integraciones B2B, ya que suelen ser objetivos fáciles para automatizaciones maliciosas si no cuentan con mecanismos de defensa adecuados.

API7:2023 Falsificación de solicitud del lado del servidor

- Restringir el acceso a recursos externos, asegurando que el sistema solo consulte dominios, puertos, esquemas y tipos de contenido previamente autorizados mediante listas de permitidos.
- Evitar comportamientos inseguros en las solicitudes, como redirecciones automáticas, y utilizar analizadores de URL confiables para prevenir interpretaciones incorrectas.
- Validar y depurar toda la información recibida, evitando procesar datos de entrada que puedan contener contenido malicioso.
- Controlar la salida de la API, enviando solo respuestas procesadas y verificadas, no contenido sin filtrar.

API8:2023 Configuración incorrecta de seguridad

Para reducir la exposición a fallas de configuración, el ciclo de vida de una API debería incluir:

- Un proceso de endurecimiento consistente que permita desplegar entornos seguros de forma rápida y siguiendo pasos repetibles.
- Revisiones periódicas de configuración en toda la pila de la API, incluyendo archivos de orquestación, componentes internos y servicios en la nube como permisos de almacenamiento.
- Evaluaciones automáticas y continuas que verifiquen si las configuraciones siguen siendo efectivas y coherentes en todos los entornos.
- Uniformidad y control en las respuestas: asegurar que todos los servidores procesen solicitudes del mismo modo y que las respuestas —incluyendo errores— sigan esquemas definidos para evitar revelar información sensible.

API9:2023 Gestión inadecuada del inventario

- Mantener un inventario claro de las APIs y servicios involucrados, registrando información clave como entorno (producción, pruebas, desarrollo), nivel de acceso permitido y los datos que intercambian.
- Documentar de manera completa el funcionamiento de la API, incluyendo autenticación, errores, límites de uso, CORS, endpoints y el comportamiento esperado de solicitudes y respuestas, preferiblemente usando herramientas automáticas dentro del flujo CI/CD.

- Controlar el acceso a la documentación y a las versiones de la API, asegurando que solo usuarios autorizados puedan consultarla, y aplicar medidas de protección externas para todas las versiones expuestas, no solo para la producción.
- Evitar el uso de datos reales fuera del entorno de producción y evaluar los riesgos cuando existan versiones antiguas, aplicando mejoras de seguridad o retirándolas si ya no cumplen con los requisitos actuales

API10:2023 Consumo inseguro de API

- Evaluar la seguridad del proveedor: Antes de usar servicios externos, es fundamental revisar su postura de seguridad y garantizar que cumplan con estándares confiables.
- Asegurar la comunicación: Todas las interacciones con APIs deben realizarse mediante canales seguros, como TLS/HTTPS, para proteger los datos en tránsito.
- Validación y saneamiento de datos: Los datos recibidos de APIs integradas deben ser siempre verificados y desinfectados antes de su uso, evitando riesgos de inyección o manipulación.

6.4. Indicadores de Mejora y Pruebas de Robustez Operativa

Con el fin de evaluar la efectividad de las medidas de seguridad y la resiliencia de la API, se definen indicadores clave de desempeño (KPIs):

- Reducción de vulnerabilidades críticas: lograr una disminución del 90% de hallazgos críticos y altos en pruebas posteriores.
- Tiempo medio de resolución (MTTR): remediar vulnerabilidades críticas en un plazo no mayor a 15 días.

- Disponibilidad del servicio bajo carga: mantener un 99.9% de disponibilidad durante pruebas de estrés con 1,000 usuarios concurrentes.
- Efectividad de controles de rate limiting: rechazar solicitudes maliciosas que excedan los umbrales configurados sin comprometer la experiencia de usuarios legítimos.

Se incluirán pruebas de rendimiento (stress test y load test) mediante herramientas como JMeter y Locust, para validar la robustez de la API frente a abusos y ataques de denegación de servicio.

6.5. Referencias Bibliográficas

1. OWASP. (2023). *OWASP API Security Top 10*. https://owasp.org/API-Security
2. PTES. (2024). *Penetration Testing Execution Standard*. https://www.pentest-standard.org
3. Ley Orgánica de Protección de Datos Personales (Ecuador). (2021). https://www.defensoria.gob.ec
4. Código Orgánico Integral Penal (Ecuador). https://www.funcionjudicial.gob.ec
5. European Union. (2016). *General Data Protection Regulation (GDPR)*. https://gdpr.eu
6. Cibersafety. (s.f.). *¿Qué es Burp Suite? La herramienta de ciberseguridad*. https://cibersafety.com/que-es-burp-suite-herramienta-ciberseguridad/
7. Wallarm. (s.f.). *OWASP ZAP: Proxy de Ataque Zed*. https://lab.wallarm.com/what/owasp-zap-proxy-de-ataque-zed/?lang=es
8. **Linode.** (2022, 11 de marzo). *API design best practices: Guía para diseñar APIs REST mantenibles y legibles, centradas en recursos*. Linode Docs. REST API Best Practices for Design | Linode Docs
9. **Stack Overflow.** (2020, March 2). *Best practices for REST API design*. *Stack Overflow Blog*. https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/
10. OWASP. (s. f.). *Key Management Cheat Sheet*. OWASP Cheat Sheet Series. Recuperado el 26 de noviembre de 2025, de https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html

11. OWASP. (2023). *API1:2023 Broken Object Level Authorization*. OWASP API Security Top 10. https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/
12. OWASP Foundation. (2025). *Server Side Request Forgery (SSRF)*. OWASP. https://owasp.org/www-community/attacks/Server_Side_Request_Forgery
13. Salt Security. (2024, August 23). *The hidden dangers of zombie and shadow APIs—and why only Salt Security can tackle them*. Salt Security. https://salt.security/blog/the-hidden-dangers-of-zombie-and-shadow-apis--and-why-only-salt-security-can-tackle-them
14. Pan, L., Cohney, S., Murray, T., & Pham, V.-T. (2023, January 23). *EDEFuzz: A web API fuzzer for excessive data exposures* (arXiv:2301.09258) [Preprint]. arXiv. https://arxiv.org/abs/2301.09258
15. Akhawe, D., Amann, B., de los Santos, D., & Steiner, M. (2020). *Securing the Modern API Surface*. Google Security Blog. https://security.googleblog.com
16. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation). University of California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
17. Fielding, R. T., & Taylor, R. N. (2000). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2), 115–150. https://doi.org/10.1145/514183.514185
18. Gómez, J., Rojas, D., & Crespo, P. (2021). Security Analysis of RESTful APIs in Cloud Environments. *IEEE Access*, 9, 55431–55445. https://doi.org/10.1109/ACCESS.2021.3070903
19. NIST. (2012). *SP 800-30: Guide for Conducting Risk Assessments*. https://doi.org/10.6028/NIST.SP.800-30r1
20. NIST. (2018). *Framework for Improving Critical Infrastructure Cybersecurity (Version 1.1)*. https://www.nist.gov/cyberframework
21. OWASP. (2023). *OWASP API Security Top 10 2023*. https://owasp.org/API-Security
22. Richardson, L., & Amundsen, M. (2016). *RESTful Web APIs*. O'Reilly Media. https://www.oreilly.com/library/view/restful-web-apis/9781449359713/
23. Ross, R., Pillitteri, V., Dempsey, K., Riddle, M., & Guissanie, G. (2020). *NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems*. https://doi.org/10.6028/NIST.SP.800-53r5
24. Salt Security. (2024). *API Security Trends Report*. https://salt.security

25. NIST. (2018). *Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1*. https://www.nist.gov/cyberframework

26. NIST. (2020). *SP 800-37 Rev. 2: Risk Management Framework*. https://doi.org/10.6028/NIST.SP.800-37r2

27. NIST. (2020). *SP 800-53 Rev. 5: Security and Privacy Controls*. https://doi.org/10.6028/NIST.SP.800-53r5

28. NIST. (2008). *SP 800-115: Technical Guide to Information Security Testing*. https://doi.org/10.6028/NIST.SP.800-115

29. OWASP Foundation. (*s.f.*). *OWASP Web Security Testing Guide (WSTG)* (Versión 4.2). The OWASP Foundation. OWASP Web Security Testing Guide | OWASP Foundation

30. OWASP Foundation. (*s.f.*). *OWASP Zed Attack Proxy (ZAP) Documentation*. The OWASP Foundation. ZAP – Documentation

Apéndices

1. Informe Owasp Zap

   https://exarjo-my.sharepoint.com/:b:/p/jorge_mena/IQDp-cnLjc8dTLf2W6r22ttQAVzGPMtmuU_7TCoR0sNUkHw?e=S8GN7H

2. Máquina virtual de la API

   https://exarjo-my.sharepoint.com/:f:/p/jorge_mena/IgA69JvMfac3RoZCO-YprHMPAfdLc4F7r7IfVD-xitt3OC4