Maestría en

# CIBERSEGURIDAD

**Trabajo previo a la obtención de título de**
**Magister en Ciberseguridad**


**AUTORES:**

Leonardo Andrés Albuja Dueñas

Luis Alfredo Guachamin Yumbo

Franklin Paúl Guamán Muzo

Deylan Javier Lema Villavicencio

**TUTORES:**

Alejandro Cortés López

Iván Reyes Chacón


**TEMA:**

## ANÁLISIS DE INTERACCIÓN DE UN PROGRAMA CON API RESTFUL BAJO ESCENARIOS DE CIBERATAQUES


Quito - Ecuador

Junio – 2025


REINVENTEMOS EL FUTURO

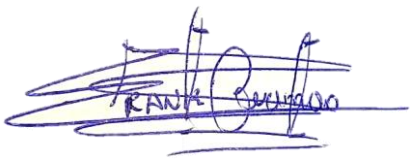
**Certificación de autoría**

Nosotros, **Leonardo Andrés Albuja Dueñas, Franklin Paúl Guamán Muzo, Luis Alfredo Guachamin Yumbo, Deylan Javier Lema Villavicencio**, 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.

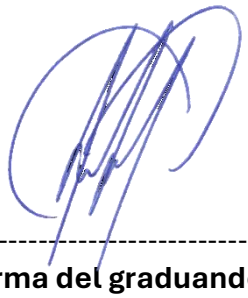
---

**Firma del graduando**
**Leonardo Andrés Albuja Dueñas**



---

**Firma del graduando**
**Franklin Paúl Guamán Muzo**



---

**Firma del graduando**
**Luis Alfredo Guachamin Yumbo**

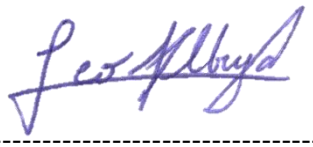
---

**Firma del graduando**
**Deylan Javier Lema Villavicencio**

**Autorización de Derechos de Propiedad Intelectual**

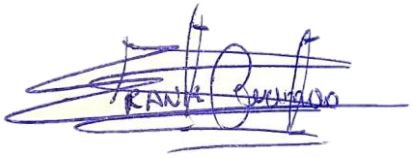
Nosotros, **Leonardo Andrés Albuja Dueñas, Franklin Paúl Guamán Muzo, Luis Alfredo Guachamin Yumbo, Deylan Javier Lema Villavicencio**, en calidad de autores del trabajo de investigación titulado ***ANÁLISIS DE UN PROGRAMA CON API BAJO ATAQUE***, autorizamos a la Universidad Internacional del Ecuador (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, (junio 2025)



---

**Firma del graduando**
**Leonardo Andrés Albuja Dueñas**



---

**Firma del graduando**
**Franklin Paúl Guamán Muzo**

---

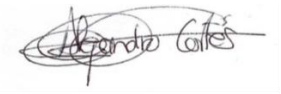
**Firma del graduando**
**Luis Alfredo Guachamin Yumbo**

---

**Firma del graduando**
**Deylan Javier Lema Villavicencio**

**Aprobación de dirección y coordinación del programa**

Nosotros, **Alejandro Cortés** e **Iván Reyes,** declaramos que: **Leonardo Andrés Albuja Dueñas, Franklin Paúl Guamán Muzo, Luis Alfredo Guachamin Yumbo, Deylan Javier Lema Villavicencio** 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**

Dedico este proyecto a mi familia, quienes siempre me han apoyado e impulsado a alcanzar mis sueños y metas. Agradezco profundamente su constante respaldo en cada etapa de mi formación.

De manera especial, dedico este logro a mi papá, por su esfuerzo incondicional y por estar presente durante todo este proceso académico. Su apoyo ha sido fundamental para llegar hasta aquí.

También extiendo esta dedicatoria a mis amigos y compañeros, quienes formaron parte de este camino y fueron un valioso apoyo a lo largo del trayecto.

## AGRADECIMIENTOS

Expreso mi más sincero agradecimiento a Dios, por darme la oportunidad de superarme una vez más, tanto en lo profesional como en lo personal.

Agradezco profundamente a mi familia, por respaldar mis deseos y metas, y por alentarme a seguir adelante en cada paso de este camino.

A mis maestros, gracias por compartir sus conocimientos y por la dedicación en cada sesión. También al personal administrativo, por su guía y apoyo constante durante todo el proceso.

A mis amigos y compañeros, gracias por ser parte de este trayecto, por su compañía, su ayuda y los momentos compartidos que lo hicieron más llevadero.

Quiero agradecer de manera muy especial a mi papá, quien ha estado a mi lado desde el principio de todo. Gracias por ser mi consejero, mi motivador y un pilar esencial en mi vida. Por darme fuerzas cuando sentí flaquear, por alentarme cuando dudé de mí mismo, y por creer en mí incluso cuando yo no lo hacía.

Gracias por ser convicción en mis momentos de titubeo.

Gracias por no dejarme renunciar a mis sueños.

Gracias por enseñarme, con tu ejemplo, a ser un mejor hombre.

## RESUEMEN

El presente proyecto tiene como propósito analizar la interacción de una API comprometida dentro de un entorno controlado de ciberataques. Para ello, se implementó un laboratorio virtual con un esquema cliente-servidor, donde se llevaron a cabo diversos ataques a través de la API de una aplicación web. Se documentó cada fase del proceso, incluyendo la ejecución de los ataques y el análisis de los resultados obtenidos.

Durante el desarrollo se utilizaron principalmente las herramientas Postman y Burp Suite, debido a sus capacidades de análisis, pruebas y explotación de vulnerabilidades en aplicaciones web. Se ejecutaron ataques como el crackeo de tokens, inyecciones NoSQL y otros tipos de explotación orientados a APIs.

Como resultado, se resalta la importancia del rol que desempeñan las APIs en la arquitectura de aplicaciones modernas y la limitada atención que suelen recibir en términos de ciberseguridad. Finalmente, se proponen recomendaciones y acciones preventivas basadas en las vulnerabilidades identificadas a lo largo del proyecto.

**Palabras clave:** API, ciberseguridad, laboratorio virtual, Burp Suite, Postman, inyección NoSQL

**ABSTRACT**

This project aims to analyze the interaction of a compromised API within a controlled cyberattack environment. To achieve this, a virtual laboratory was set up using a client-server architecture, where various attacks were executed through the API of a web application. Each stage of the process was documented, including the execution of attacks and the analysis of the results obtained.

The tools primarily used during the development were Postman and Burp Suite, due to their capabilities in analysis, testing, and exploitation of web application vulnerabilities. Attacks such as token cracking, NoSQL injection, and other API-focused exploitation techniques were performed.

The results highlight the critical role that APIs play in modern application architecture and the limited attention they often receive in terms of cybersecurity. Finally, the project presents recommendations and preventive actions based on the vulnerabilities identified throughout the research.

**Keywords:** API, cybersecurity, virtual lab, Burp Suite, Postman, NoSQL injection

**TABLA DE CONTENIDOS**

I. INTRODUCCION ..... 12
II. GLOSARIO ..... 16
III. OBJETIVO Y ALCANCE ..... 19
IV. METODOLOGIA ..... 20
4.1. INSTALACIÓN DE DOCKER ..... 20
4.2. CONFIGURACIÓN Y ARRANQUE DE APLICACIÓN CRAPI ..... 21
4.3. FUNCIONALIDADES DE CRAPI ..... 24
4.4. ANÁLISIS DE FUNCIONAMIENTO DE API ..... 29
4.5. GENERACIÓN DE TOKENS CON JWT TOOL ..... 38
4.6. CRACKEO DE TOKEN ..... 47
4.7. ATAQUE BOLA (BROKEN OBJECT LEVEL AUTHORIZATION) ..... 52
4.8. ATAQUE DE INYECCION SQL ..... 57
V. RESULTADOS ..... 64
5.1. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR CRACKEO DE JSON WEB TOKEN (JWT) ..... 65
5.2. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR ATAQUE BOLA ..... 72
5.3. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR ATAQUES DE INYECCIÓN SQL EN APIS ..... 77
VI. ANEXOS ..... 85
VII. REFERENCIAS ..... 86

**RESUMEN DE FIGURAS**

**Figura 1** VM de crAPI ..... 20
**Figura 2** *Instalación de Docker* ..... 20
**Figura 3** *Habilitación de Docker* ..... 21
**Figura 4** *Inicio de servicio Docker* ..... 21
**Figura 5** *Estado de servicio Docker* ..... 21
**Figura 6** *Descarga de repositorio crAPI* ..... 21
**Figura 7** *Carpeta de configuración de crAPI* ..... 22
**Figura 8** *Configuración de permisos de ejecución* ..... 22
**Figura 9** *Ejecución de Shell script* ..... 22
**Figura 10** *Arranque de aplicación crAPI* ..... 22
**Figura 11** *Aplicación crAPI en ejecución* ..... 23
**Figura 12** *Acceso a aplicación crAPI* ..... 23
**Figura 13** *Acceso a servicio de correos de crAPI* ..... 24
**Figura 14** *Creación de usuario* ..... 25
**Figura 15** *Registro exitoso de usuario* ..... 25
**Figura 16** *Inicio de sesión test1* ..... 25
**Figura 17** *Inicio de sesión user2* ..... 26
**Figura 18** *Inicio de sesión user3* ..... 26
**Figura 19** *Página principal* ..... 27
**Figura 20** *Revisión de correo* ..... 27
**Figura 21** *Registro de Vehículo* ..... 28
**Figura 22** *Datos de vehículo registrado* ..... 28
**Figura 23** *VM atacante* ..... 29
**Figura 24** *Extensión FoxyProxy* ..... 29
**Figura 25** *Configuración de proxys en FoxyProxy* ..... 30
**Figura 26** *Configuración de proxy en Postman* ..... 30
**Figura 27** *Inicio de captura en Postman* ..... 31
**Figura 28** *Request de login capturada* ..... 31
**Figura 29** *Token generado* ..... 32
**Figura 30** *Sección Body* ..... 32
**Figura 31** *Habilitación de custom proxy* ..... 33
**Figura 32** *Request interceptado* ..... 34
**Figura 33** *Visualización de request y response* ..... 35
**Figura 34** *Envío de token a "Decoder"* ..... 35
**Figura 35** *Sección de "Decoder"* ..... 36
**Figura 36** *Información decodificada* ..... 36
**Figura 37** *Herramienta web jwt.io* ..... 37
**Figura 38** *Token decodificado* ..... 38
**Figura 39** *Clonación de repositorio desde github* ..... 38

**Figura 40** *Habilitación de ambiente virtual* ..... 39
**Figura 41** *Activación de ambiente y ejecución de programa JWT_TOOL* ..... 39
**Figura 42** *Request interceptado* ..... 40
**Figura 43** *Token de apartado community* ..... 41
**Figura 44** *API de sección Community* ..... 41
**Figura 45** *Token original de sección Community* ..... 42
**Figura 46** *Envío de request a sección Repeater* ..... 42
**Figura 47** *Ejecución de jwt_tool* ..... 43
**Figura 48** *Opciones por apartado de Exploit* ..... 43
**Figura 49** *Tokens genéricos* ..... 43
**Figura 50** *Token original* ..... 44
**Figura 51** *Reemplazo por token genérico* ..... 45
**Figura 52** *Mensaje de error* ..... 45
**Figura 53** *Generación de token sin firma* ..... 46
**Figura 54** *Envío de request con token sin firma* ..... 46
**Figura 55** *Creación de diccionario* ..... 47
**Figura 56** *Visualización de diccionario* ..... 48
**Figura 57** *Intercepción de API por dashboard* ..... 48
**Figura 58** *Ejecución de jwt_tool por cracking* ..... 49
**Figura 59** *Hallazgo de clave de firma* ..... 49
**Figura 60** *Estructura de token original* ..... 50
**Figura 61** *Estructura de token alterado* ..... 50
**Figura 62** *Reemplazo de token alterado* ..... 51
**Figura 63** *Response con token alterado* ..... 51
**Figura 64** *Intercepción de API desde navegador* ..... 53
**Figura 65** *Envío de request a postman* ..... 53
**Figura 66** *Visualización de información del request* ..... 54
**Figura 67** *Visualización de información del request 2* ..... 54
**Figura 68** *Request de vehículo* ..... 55
**Figura 69** *Información de vehicleid* ..... 56
**Figura 70** *Alteración en estructura de request* ..... 56
**Figura 71** *Información de user2* ..... 57
**Figura 72** *Sección de "Shop" de la App web* ..... 58
**Figura 73** *Intercepción de request por validar cupón* ..... 58
**Figura 74** *Envío de request a "Intruder"* ..... 59
**Figura 75** *Payload noSQL* ..... 60
**Figura 76** *Payload SQL* ..... 60
**Figura 77** *Carga de Payload* ..... 61
**Figura 78** *Opción de codificar payload* ..... 61
**Figura 79** *Inicio de ataque* ..... 62
**Figura 80** *Sentencias exitosas* ..... 62

**Figura 81** *Visualización de cupón* ..... 63

## RESUMEN DE TABLAS

**Tabla 1** *Ejemplo de Endpoint para obtener la información de un usuario por ID*... 74
**Tabla 2** *Ciclo de ataque típico de una inyección SQL en APIs.* ..... 78
**Tabla 3** *Factores que propician la vulnerabilidad.* ..... 80
**Tabla 4** *Estrategias preventivas contra la inyección SQL en APIs.* ..... 80
**Tabla 5** *Herramientas de seguridad útiles.* ..... 82
**Tabla 6** *Automatización en la detección de vulnerabilidades.* ..... 85

# I. INTRODUCCION

Gracias al desarrollo integración de las APIs se ha logrado optimizar tiempos y recursos en el área de desarrollo, ayudando así a muchos desarrolladores a comunicar programas entre sí, realizar transacciones, solicitar y enviar información de una manera mucho más simple y empleando menos recursos de lo que previamente se necesitaba emplear y aplicar para el desarrollo de un programa.

Es debido a este uso sin precautela y desmedido por parte de los desarrolladores de las ya nombradas APIs, que se han presentado distintas vulnerabilidades y amenazas con el pasar de tiempo. Un ciberdelincuente con conocimiento y motivos suficientes sabe que se hará uso de APIs para una sección de login, un proceso de transacción de datos, geolocalización, entre otros procesos, por las cuales el ciberdelincuente podría enfocar todas sus habilidades y esfuerzos en realizar robo de información, elevación de permisos, suplantación de identidad entre otras ciberactividades delictivas que sean de su beneficio.

Por lo tanto, es debido a este eslabón vulnerable en el mundo de la tecnología que muchas organizaciones están enfocando sus recursos en fortalecer la seguridad de las APIs, sin embargo, la seguridad de esta tecnología no es comprendida en su totalidad ya que evoluciona constantemente y a pasos agigantados. Es por esto la importancia de realizar laboratorios y ataques en entornos controlados contra este tipo de tecnologías, con la finalidad de

comprender y tomar el papel de atacante, desarrollando habilidades y aprendiendo como y donde están las fallas y puntos débiles de las APIs, permitiéndonos así contrarrestar estas amenazas y reforzar la seguridad de su estructura, solicitudes, funciones y procesos.

Como un evento destacado, se tiene el registro de un ataque por medio de API a la red social twitter, lo cual tuvo como resultado la filtración de información 5.4 millones de cuentas de usuarios en todo el mundo, conteniendo detalles como localización ¿, usuarios, nombres, twitter ID, correos electrónicos y números de teléfono. El ataque a través de API se pudo realizar ya que a través de la API de twitter se podría validar si la cuenta de un usuario tenía registrada su número de teléfono o su cuenta de correo.

Por parte de la entidad OWASP se tiene contemplado el riesgo de seguridad que implica el uso de API, y más aun con su demanda creciente y sin control por parte de desarrolladores, por tal motivo que esta entidad ha elaborado un registro con su Top 10 de riesgos principales de las APIs:

**Autorización de Nivel de Objeto Roto:** Ya que las API suelen exponer puntos finales como identificadores de objetos, se deja la puerta abierta a ataques de control de acceso por nivel de objeto. De esta manera, en caso de no tener una validación correspondiente, un atacante podría acceder a información ajena y directorios a los cuales su usuario no debería tener acceso.

**Autenticación rota:** Se basa en aprovechar la nula o incorrecta mecánica de autenticación, permitiendo así a un atacante el uso de tokens genéricos y explotar fallas en el proceso de autenticación para tomar el perfil de otros usuarios, ya sea de forma temporal o permanente.

**Autorización de Nivel de Propiedad de Objetos Rotos:** Este riesgo combina la exposición excesiva de datos y la asignación en masa, por lo que se genera la incorrecta validación de autorización a nivel de propiedad del objeto, lo que conduce a la exposición o manipulación de datos por usuarios no autorizados.

**Consumo de Recursos sin restricciones:** Para el uso de API se requiere de consumo en CPU, almacenamiento, memoria y ancho de banda. También en ciertos programas se emplean recursos como llamadas telefónicas, mensajes de texto y correos electrónicos, pagando el consumo correspondiente de estos servicios. Es por medio de estos recursos se puede generar un ataque de denegación de servicio o aumentar los costos operativos.

**Autorización de Nivel de Función Rota:** A través de políticas de control de acceso de grupos, roles o jerarquías, mal empleadas y complejas, es probable encontrar fallas de autorización, por lo que un atacante podría acceder a recursos y/o funciones administrativas de usuarios.

**Acceso sin restricciones a Flujos de Negocio Sensibles:** Esta vulnerabilidad se refiere a la API que se emplea en un flujo comercial, tal como compras online, o publicación de contenido, sin tener en cuenta el uso excesivo y de forma automatizada.

**Falsificación de Solicitud del Lado del Servidor (SSRF):** Este riesgo se presenta cuando una API realiza la búsqueda de un recurso remoto sin la validación del URI (Uniform Resource Identifier) provisto por un usuario. Esto da paso a que un atacante fuerce a la aplicación el envío de solicitudes alteradas.

**Desconfiguración de seguridad:** Se refiere a la configuración de nivel muy bajo respecto a la seguridad establecida por desarrolladores. Esto se debe a que las API y sus aplicaciones con las cuales se integran tienen configuraciones muy complejas, por lo que los desarrolladores tratan de evitar perder estas configuraciones, evitando así buenas prácticas de seguridad al momento de la configuración de API.

**Gestión de Inventario Inadecuada:** Se refiere al inventario correcto de los host y versiones de API que se implementan, con la finalidad de evitar problemas de versiones obsoletas y puntos finales expuestos. Se hace énfasis en que la documentación adecuada, y el uso de software actualizado, en nuestro caso las API, son de suma importancia.

**Consumo Inseguro de API:** Esta vulnerabilidad nace a pie de que los desarrolladores suelen confiar en la información recibida a través de API de terceros más que en la que es provista por el usuario, por lo que tienden a tomar procesos de seguridad débiles. Para comprometer las API se centran en los servicios integrados de terceros, en lugar de ir por a API de destino de forma directa.

## II. GLOSARIO

**API:** La definición de las siglas API es Interfaz de Desarrollo de Aplicaciones lo que indica que un API debe aplicar las reglas con las cuales un sistema nuevo puede comunicarse con sistemas ya existentes. Las comunicaciones se establecen sin importar las especificaciones técnicas de su implementación como pueden ser: protocolos de comunicación, lenguajes de programación, uso de frameworks o microframeworks, arreglo de servidores, especificación de módulos, entre otras. La API debe estar bien estructurada para que sea posible implementarla con cualquier tecnología. (Valdez Beas, 2020)

**REST:** Las interfaces REST se definen únicamente en Identificadores Uniformes de Recursos (URI) para la detección e interacción con recursos, especialmente con el Protocolo de Transferencia de Hipertexto (HTTP) parala transferencia de información. Un Uri únicamente proporciona ubicación y nombre del recurso, el cual actúa como un identificador único. Los verbos predefinidos se

utilizan para definir el tipo de operación que el recurso debe realizar (Get para recuperar, Delete para eliminar, Post para crear, Update actualizar). (Neumann et al., 2021)

**API RESTful:** Es aquella API que cumple con la arquitectura REST. Basa su comunicación en HTTP. Realiza intercambio de información asegurando seguridad, confiabilidad y eficiencia. Carecen de estado por lo que se pueden realizar llamadas individuales y cada una contendrá datos necesarios para llevar a cabo el request correctamente. Una API RESTful envía una representación del recurso al usuario cuando se realiza una petición de cliente. Esta representación está estructurada en diferentes tipos de datos, entre algunos: JSON (Notación de Objetos JavaScript), HTML y texto plano que se pueden enviar mediante HTTP. JSON es el formato de archivo más utilizado, ya que es independiente del lenguaje y comprensible tanto para hu8manos como máquinas. (Golmohammadi et al., 2024)

**Método GET:** Es la petición empleada para obtener un recurso de un servidor. El servidor localiza la información requerida y la envía de vuelta al cliente. Una petición GET lleva a cabo una función de lectura y es el método de solicitud que se utiliza de manera estándar. (Erinjeri, 2022)

**Método POST:** Es la petición empleada para generar un nuevo elemento en un servidor. El servidor crea un nuevo registro en el recurso y notifica si la

generación se llevó a cabo con éxito. Una solicitud de POST ejecuta una acción de creación. (Erinjeri, 2022)

**Método PUT/PATCH:** Ambas peticiones sirven para modificar un recurso en un servidor y comunicar el estado de dicha solicitud. La principal distinción es que PUT se usa para sustituir completamente le recurso, mientras que PATCH se emplea para realizar cambios parciales. Tanto PUT como PATCH forman parte de la acción de actualizar. (Erinjeri, 2022)

**Método DELETE:** Petición empleada para eliminar el recurso en un servidor. El servidor quita una sección del recurso y notifica si la supresión fue exitosa. Una petición DELETE ejecuta una operación de eliminación. (Erinjeri, 2022)

**Burpsuite:** Es un software que contiene distintas herramientas para realizar ataque de vulnerabilidades web. Tiene funcionalidades como proxy, escáner de vulnerabilidades, editor de solicitudes entre otras funcionalidades que permiten al usuario interceptar, analizar y alterar el tráfico web

**Postman:** Es aquella plataforma que permite a los usuarios gestionar, colaborar y testear APIs. Mayormente es utilizada por desarrolladores. Ayuda a probar el funcionamiento de servicios web y aplicaciones.

## III. OBJETIVO Y ALCANCE

Como objetivo de este proyecto se prevé analizar el comportamiento, rendimiento y comunicación de una API RESTful que interactúa con un programa, comprendiendo la información generada por las solicitudes y respuestas de la API. Se propone poner dicha API en un escenario controlado de ciberataque, con la finalidad de documentar y analizar los resultados obtenidos.

Como alcance del proyecto, se planifica levantar un ambiente controlado conformado por máquinas virtuales, las cuales tomaran papeles relevantes a lo largo del proyecto, tales como host, interceptador de solicitudes de APIs, atacante, servidor web.

Se establece el uso de herramientas tales como Burp Suite y Postman para el análisis y ataque orientado a las solicitudes que se solicitaran y se recibirán de parte de la API que se tenga como objetivo.

Se empleará el uso de la aplicación web crAPI la cual significa “Completely Ridiculous API” para realizar el análisis de solicitudes y vulnerabilidades. Esta aplicación está diseñada directamente para el aprendizaje y comprensión de vulnerabilidades de API conectadas a un programa.

# IV. METODOLOGIA

## 4.1. INSTALACIÓN DE DOCKER

Para iniciar la práctica propuesta y el análisis correspondiente de la interacción entre una API y un programa, deberemos preparar una VM la cual será la encargada de correr la aplicación crAPI, esto a través de un contenedor en Docker, por lo cual deberemos instalar el servicio de Docker. El servicio de crAPI correrá en una VM con Kali Linux, dicha VM tomará el papel de servidor a lo largo del proyecto. También es importante indicar que todas las VM del proyecto tendrán sus interfaces de red en bridge para mantener una comunicación directa entre todos los equipos, incluso el equipo físico.

**Figura 1**

*VM de crAPI*

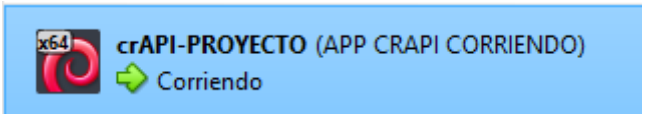



**Figura 2**

*Instalación de Docker*

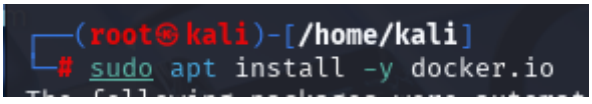


**Figura 3**

*Habilitación de Docker*

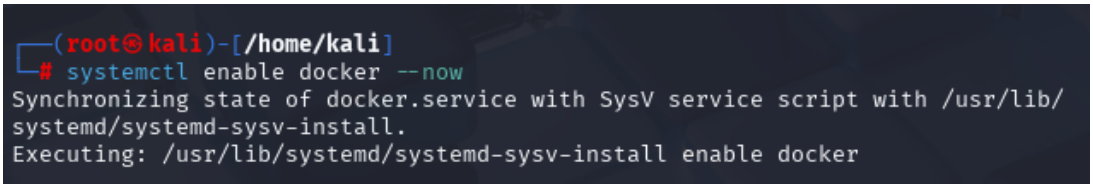



**Figura 4**

*Inicio de servicio Docker*

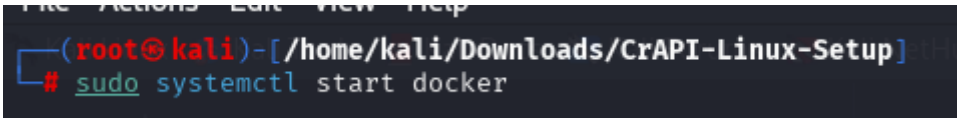
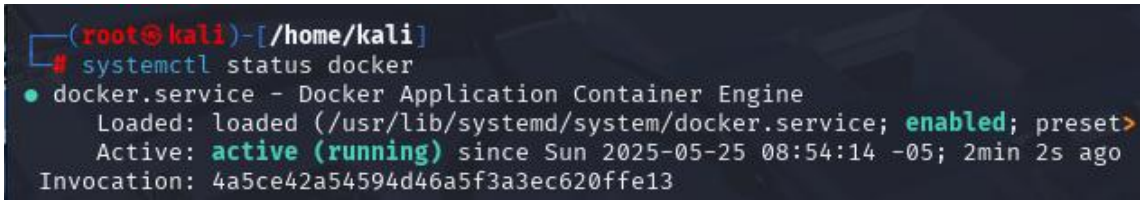



**Figura 5**

*Estado de servicio Docker*




Una vez el servicio de Docker se haya instalado en nuestra VM, y ya se encuentre corriendo, procederemos a levantar la aplicación web desde un contenedor preconfigurado.

## 4.2. CONFIGURACIÓN Y ARRANQUE DE APLICACIÓN CRAPI

**Figura 6**

*Descarga de repositorio crAPI*

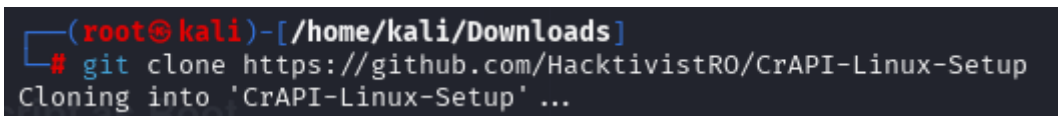


**Figura 7**

*Carpeta de configuración de crAPI*


**Figura 8**

*Configuración de permisos de ejecución*


**Figura 9**

*Ejecución de Shell script*


**Figura 10**

*Arranque de aplicación crAPI*

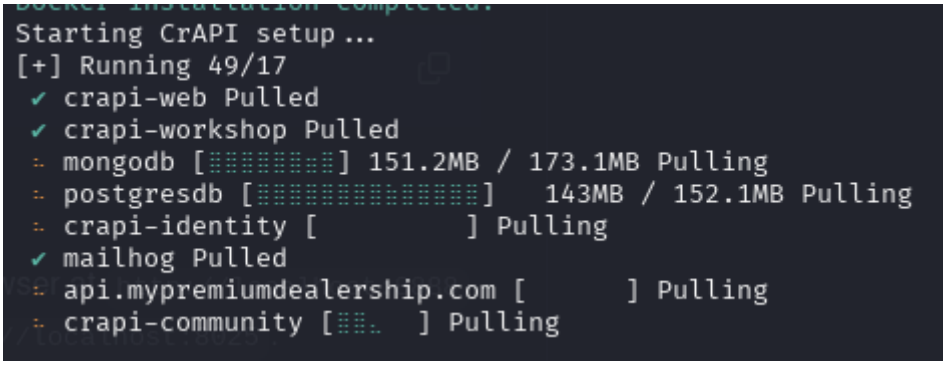


**Figura 11**

*Aplicación crAPI en ejecución*

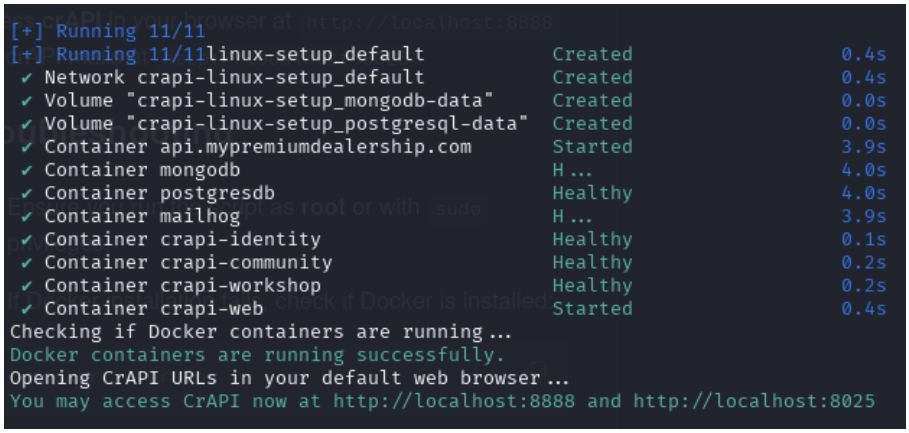



**Figura 12**

*Acceso a aplicación crAPI*

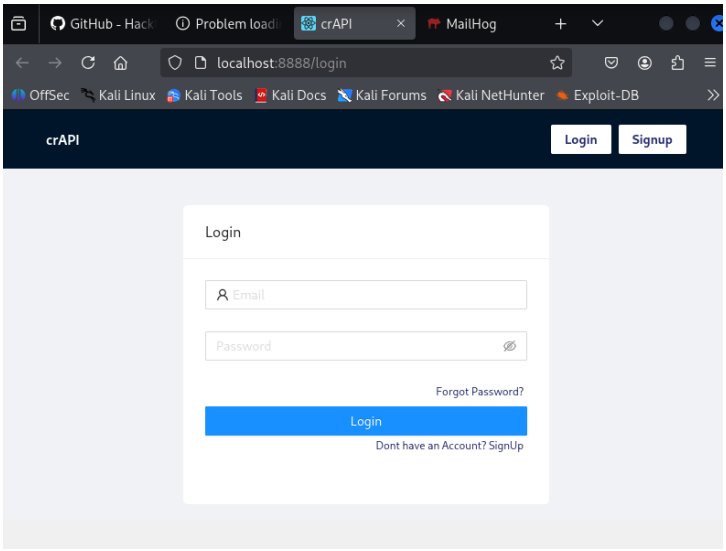
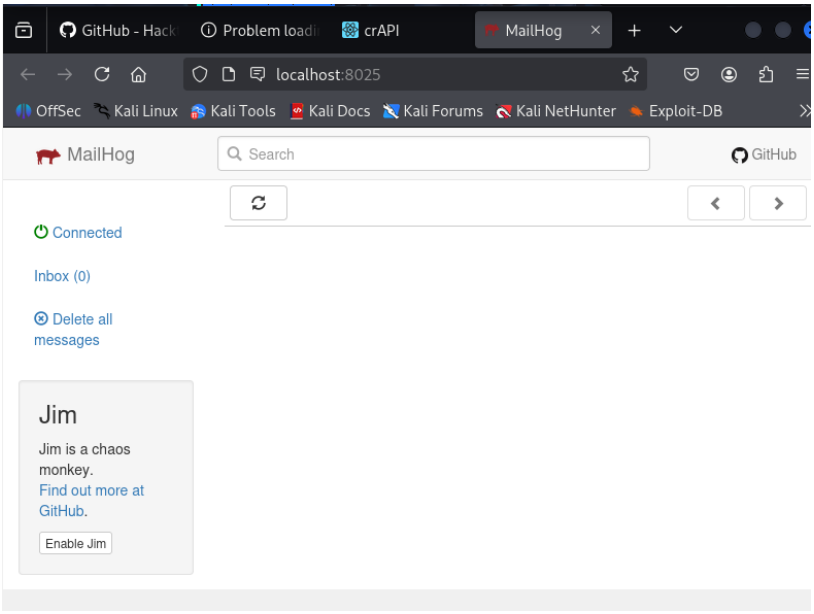


**Figura 13**

*Acceso a servicio de correos de crAPI*




## 4.3. FUNCIONALIDADES DE CRAPI

crAPI es una aplicación web que simula el registro de vehículos, consulta de ubicación en tiempo real, consulta de detalles del vehículo, información del usuario propietario del vehículo, entre otras funciones. Respectivamente la app cuenta con la creación de usuarios, recuperación de contraseña, sección de login y servicio de correo electrónico. Procederemos a utilizar estas funciones para observar y comprender su funcionamiento a un nivel general.

**Figura 14**

*Creación de usuario*

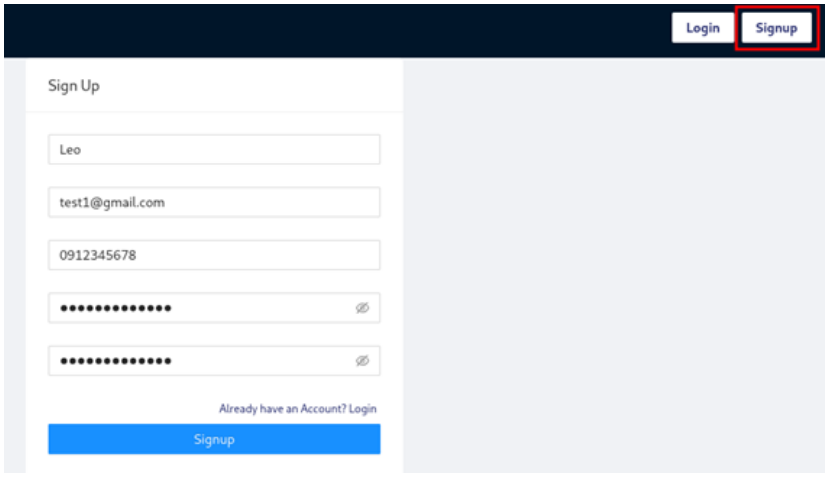



**Figura 15**

*Registro exitoso de usuario*

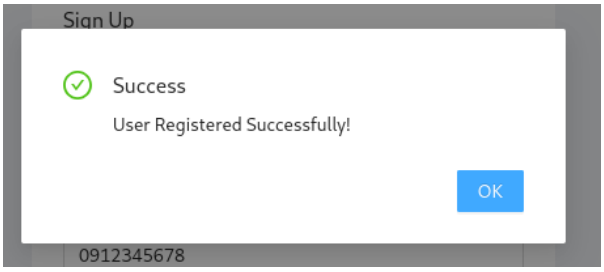



**Figura 16**

*Inicio de sesión test1*

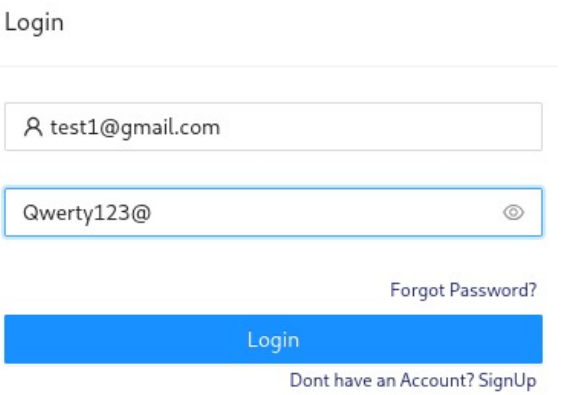


**Figura 17**

*Inicio de sesión user2*

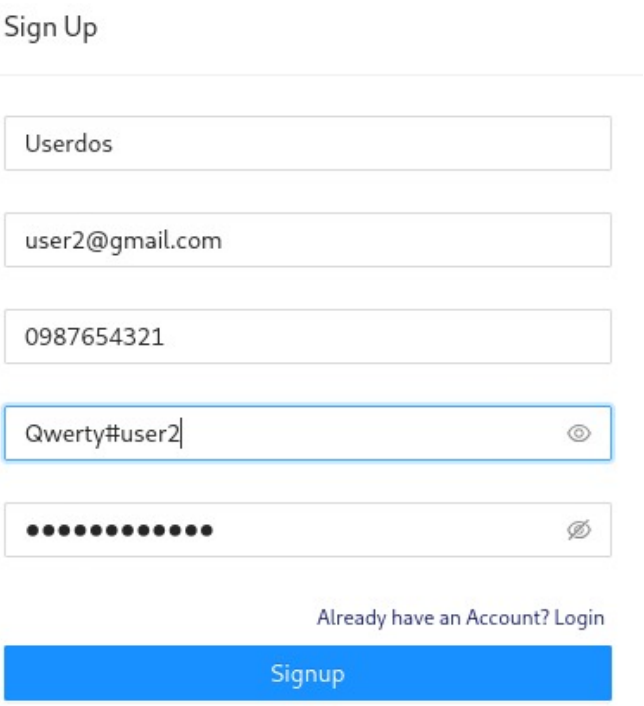



**Figura 18**

*Inicio de sesión user3*

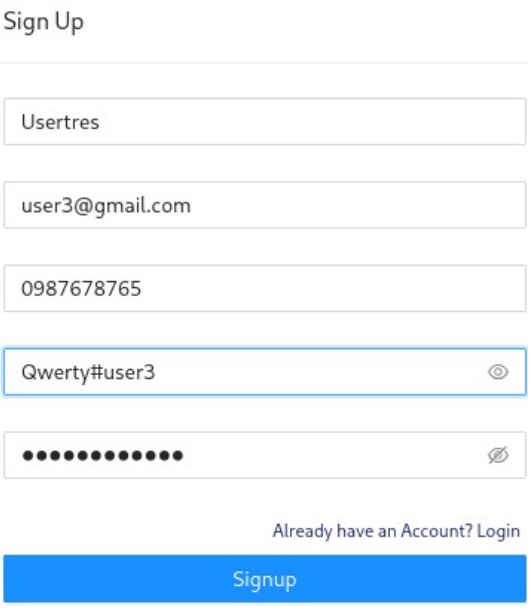
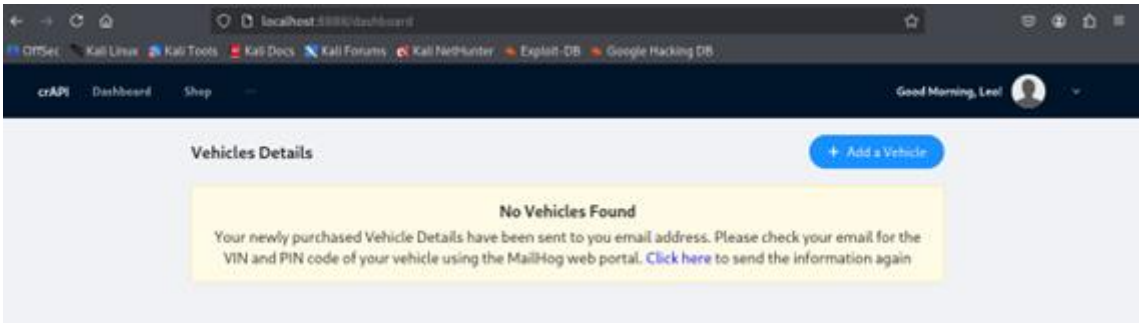


**Figura 19**

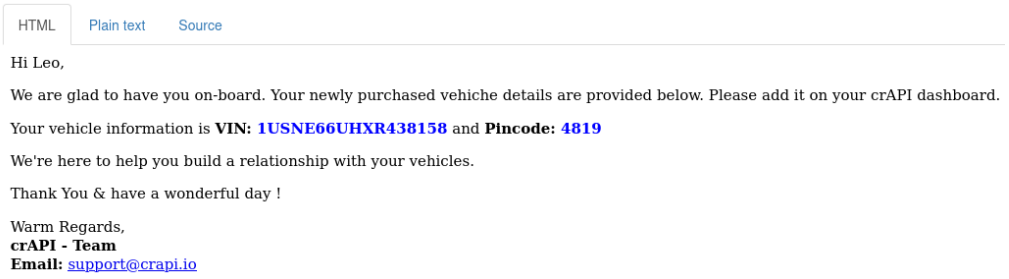
*Página principal*




En este punto, la aplicación nos indica para reenviar el token para añadir un vehículo se debe dar clic al hipervínculo, y revisar nuestro correo para validar dicha información.

**Figura 20**

*Revisión de correo*




Con la información ya a la mano, podemos continuar con el proceso de “Add a Vehicle” dando clic al botón correspondiente. Nos enviará a una página donde pedirá la información provista en el correo y esta será validada, validando al final que el vehículo a registrar se guardó de manera exitosa.

**Figura 21**

*Registro de Vehículo*

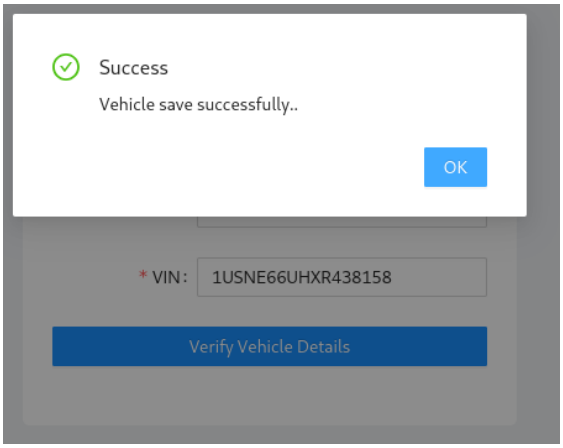
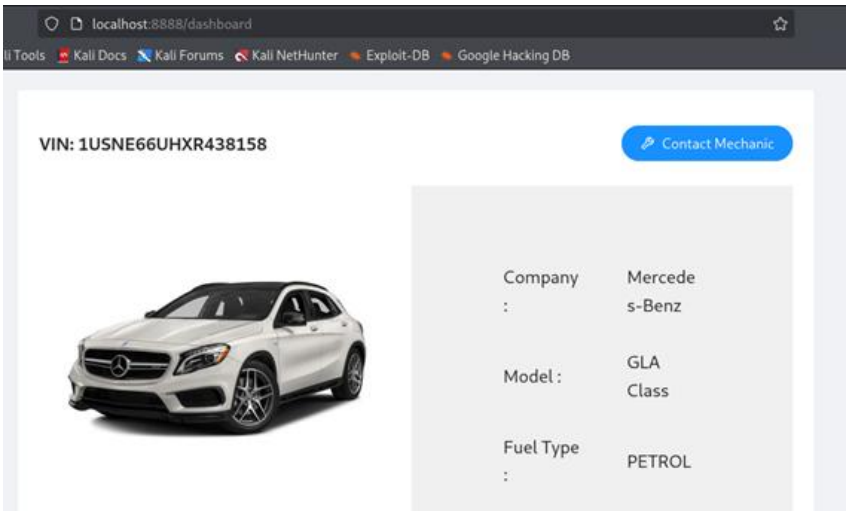



**Figura 22**

*Datos de vehículo registrado*



### 4.4. ANÁLISIS DE FUNCIONAMIENTO DE API

Con la aplicación web operativa, y validada su conexión en nuestra red local, accederemos desde un equipo atacante a la app web, capturando así sus solicitudes a través de la aplicación Postman dando así inicio al análisis de la API.

**Figura 23**

*VM atacante*

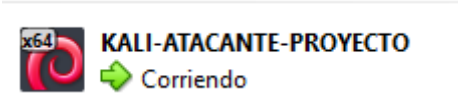



**Figura 24**

*Extensión FoxyProxy*

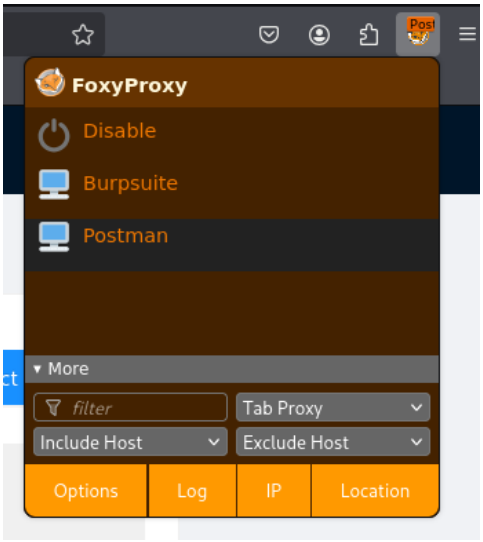
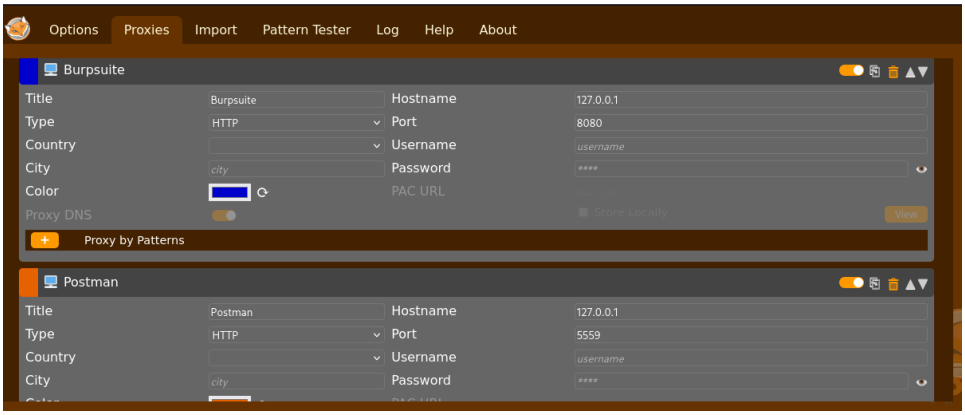


**Figura 25**

*Configuración de proxys en FoxyProxy*




Con la ayuda de la extensión FoxyProxy para Mozilla podremos administrar de manera rápida y sencilla los proxys de las herramientas para realizar la práctica. En la herramienta Postman debemos de iniciar el servicio de proxy permitiéndonos así capturar en tiempo real las solicitudes enviadas y recibidas.

**Figura 26**

*Configuración de proxy en Postman*

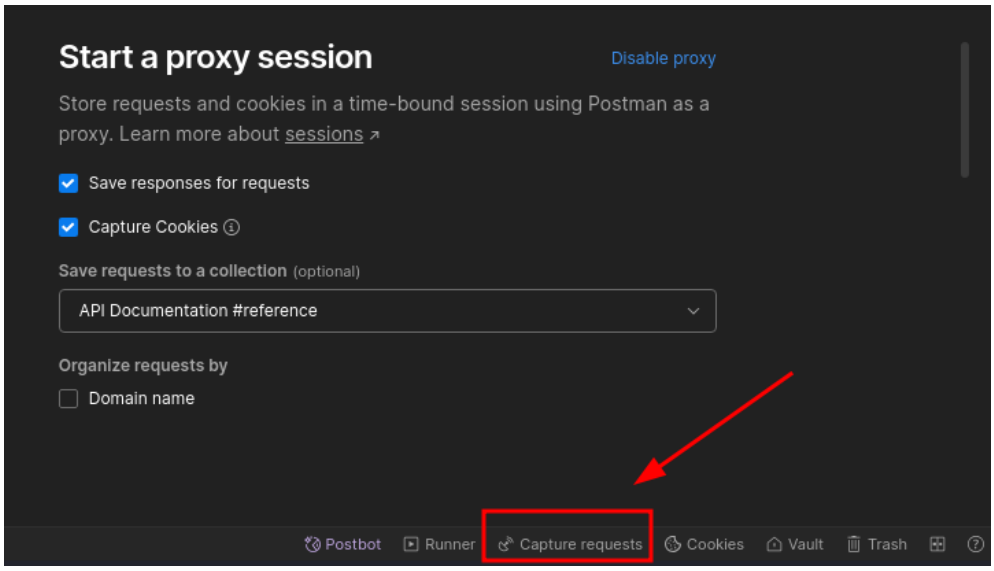
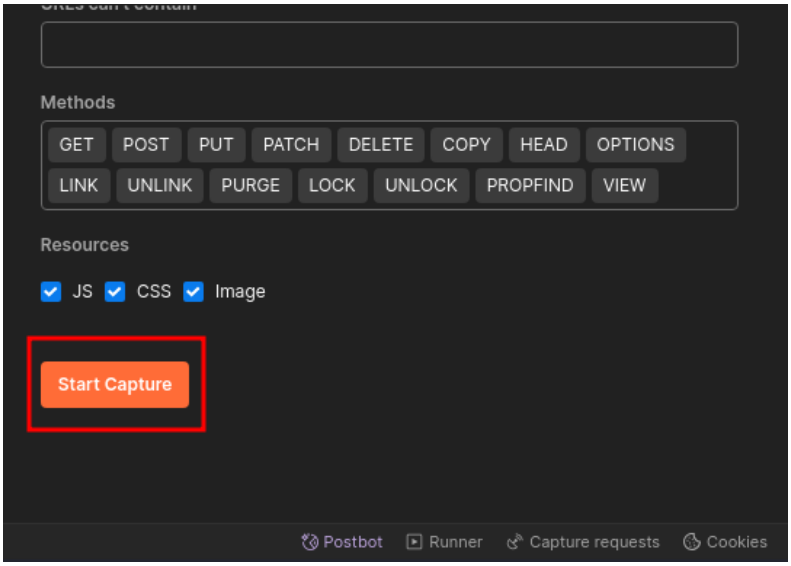


**Figura 27**

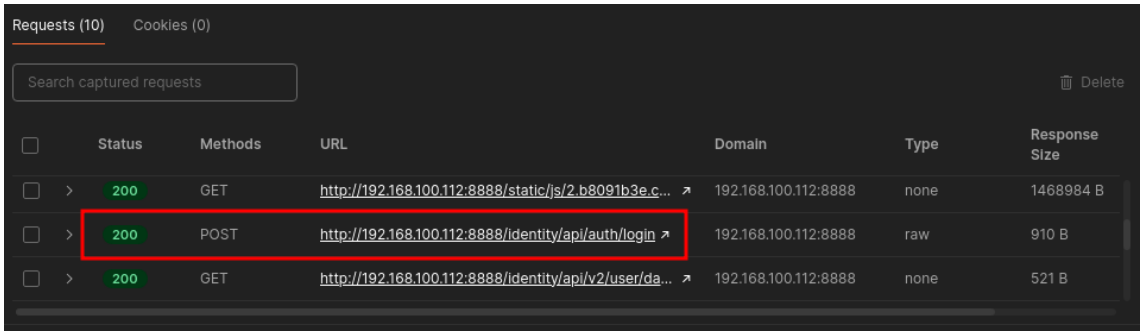
*Inicio de captura en Postman*




Iniciaremos sesión desde la aplicación web, y capturaremos el request de inicio de sesión, esto lo diferenciaremos por el método POST y el apartado de login. Sobre esta solicitud realizaremos el análisis inicial.

**Figura 28**

*Request de login capturada*



Procederemos a trabajar directamente sobre la request de login, validando así el token que ha generado la API, e incluso se puede visualizar el usuario y contraseña de manera legible en el aparatado de Body.

**Figura 29**

*Token generado*

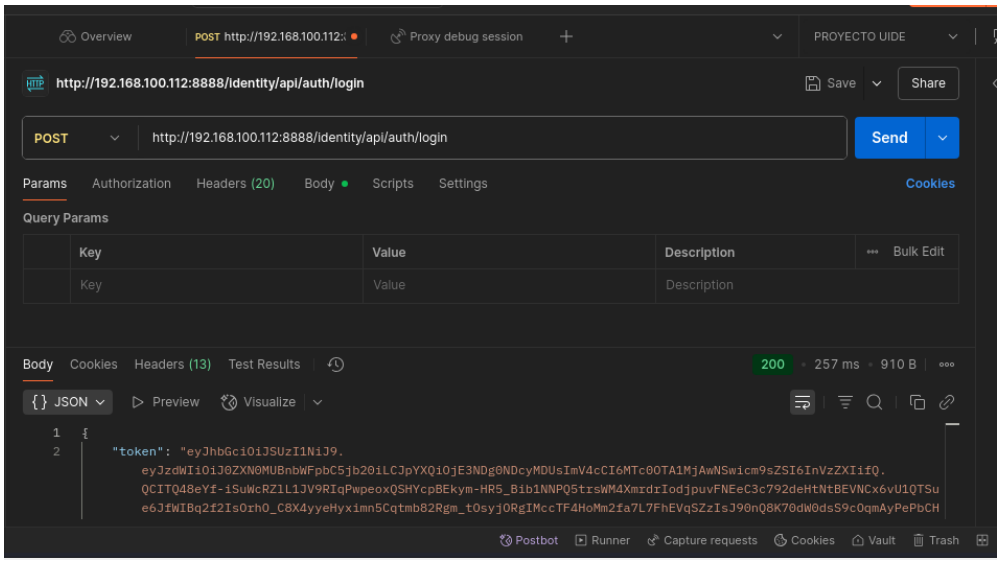
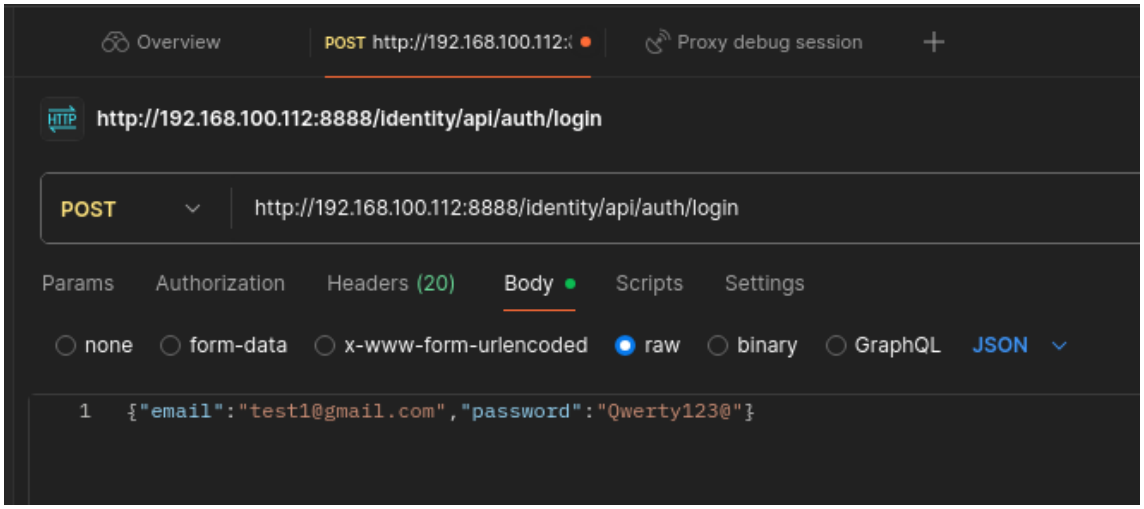



**Figura 30**

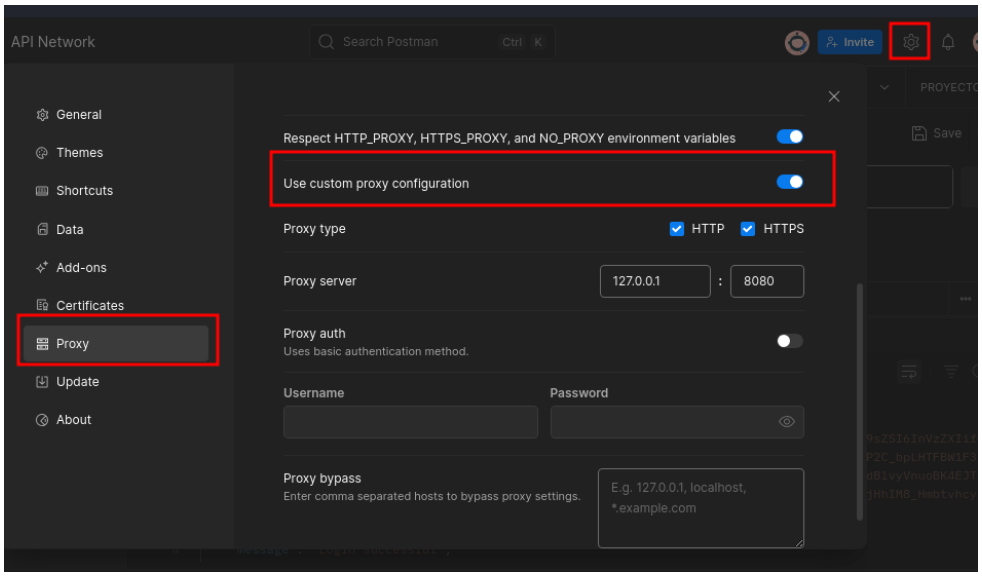
*Sección Body*



Con los datos obtenidos, integraremos el uso de Postman junto con Burpsuite. Para hacer esto debemos habilitar el proxy personalizable en Postman, desde su apartado de Settings.

**Figura 31**

*Habilitación de custom proxy*



Una vez configurado el proxy en Postman, enviaremos nuevamente el request capturado, y validaremos en Burpsuite su captura.

**Figura 32**

*Request interceptado*

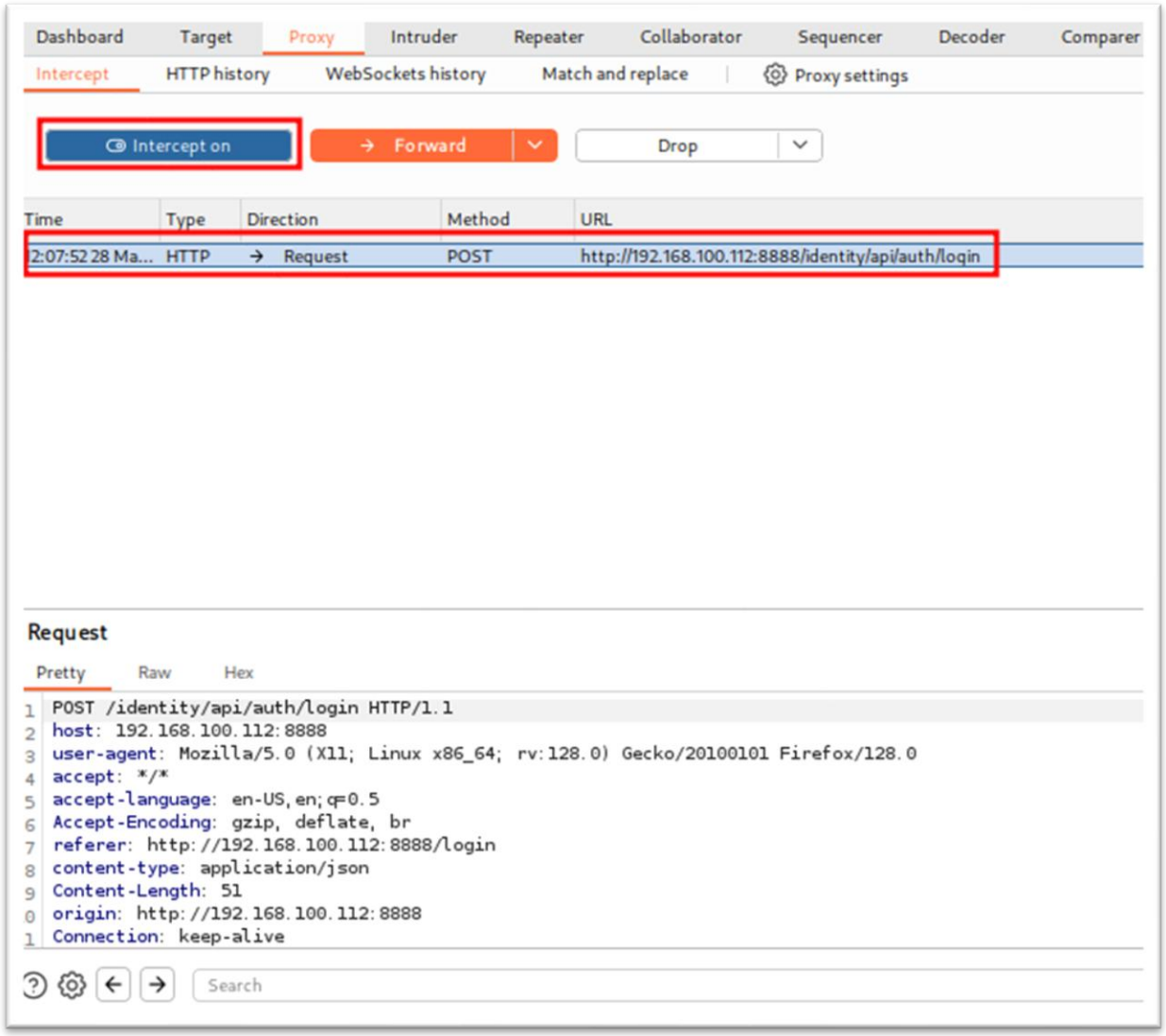
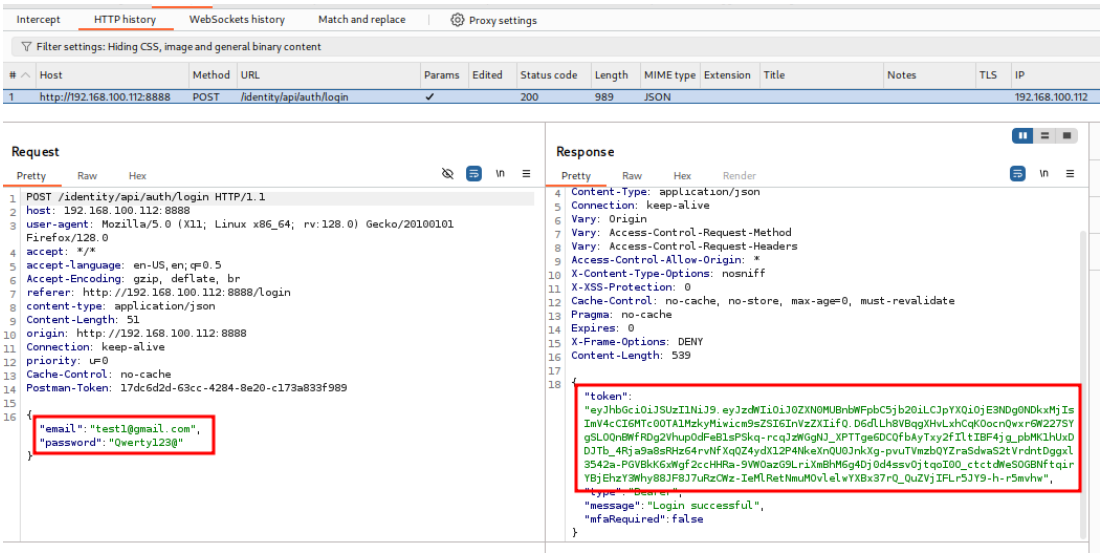


**Figura 33**

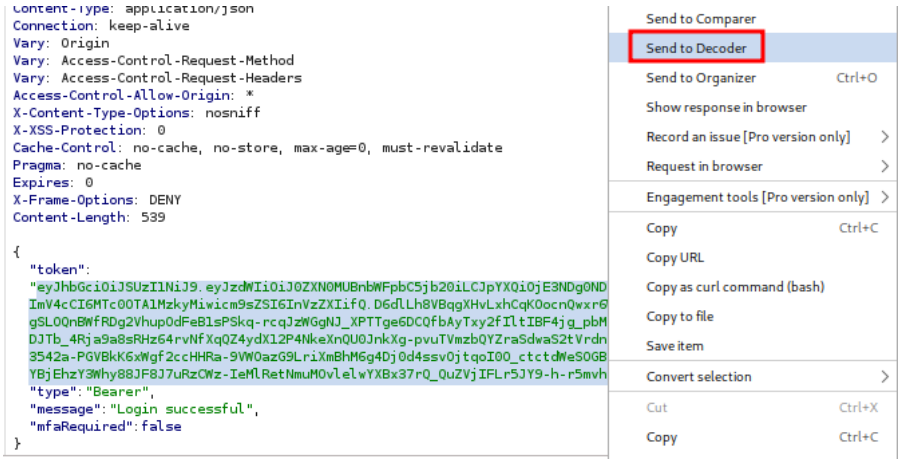
*Visualización de request y response*




Habiendo capturado el token generado por la API, podremos decodificarlo en la herramienta Burpsuite enviándolo a la sección de “Decoder” para visualizar su información y contenido.

**Figura 34**

*Envío de token a "Decoder"*



Con la información del token enviada en la sección de Decoder, deberemos seleccionar entre las opciones de “Decode as” y seleccionar “Base64” debido a que cada token está conformado por tres partes importantes, y cada parte esta codificada en base64.

**Figura 35**

*Sección de "Decoder"*

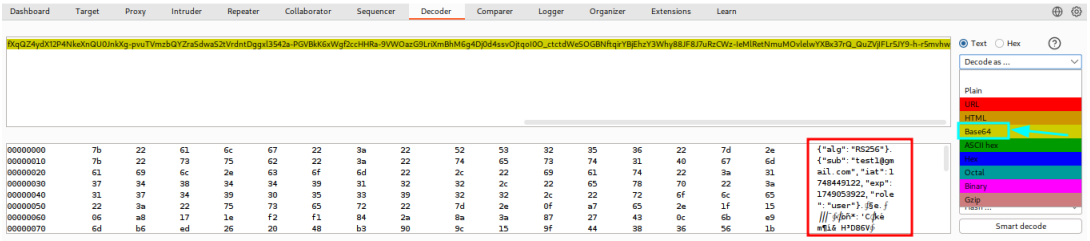
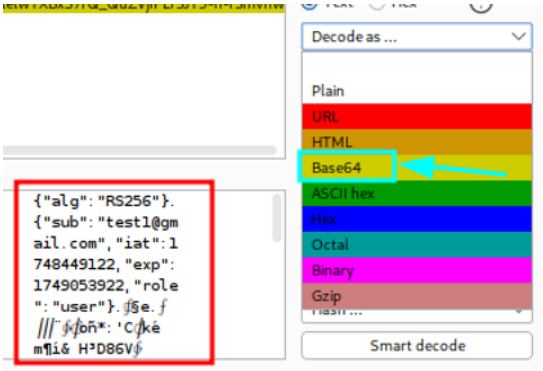



**Figura 36**

*Información decodificada*




De esta manera podemos visualizar y comprender que el token contiene toda la información enviada por parte del usuario, validando así el usuario, firma, body. Tomar en cuenta que la estructura de un JWT (Jason Web Token) está conformada por 3 partes que son header, payload y signature.

Con la finalidad de hacer el análisis del JWT de forma más cómoda, haremos uso del decodificador de la herramienta web jwt.io, el cual tiene la misma función que el decodificador de Burpsuite, pero visualmente es más detallado con respecto a color y estructura.

**Figura 37**

*Herramienta web jwt.io*

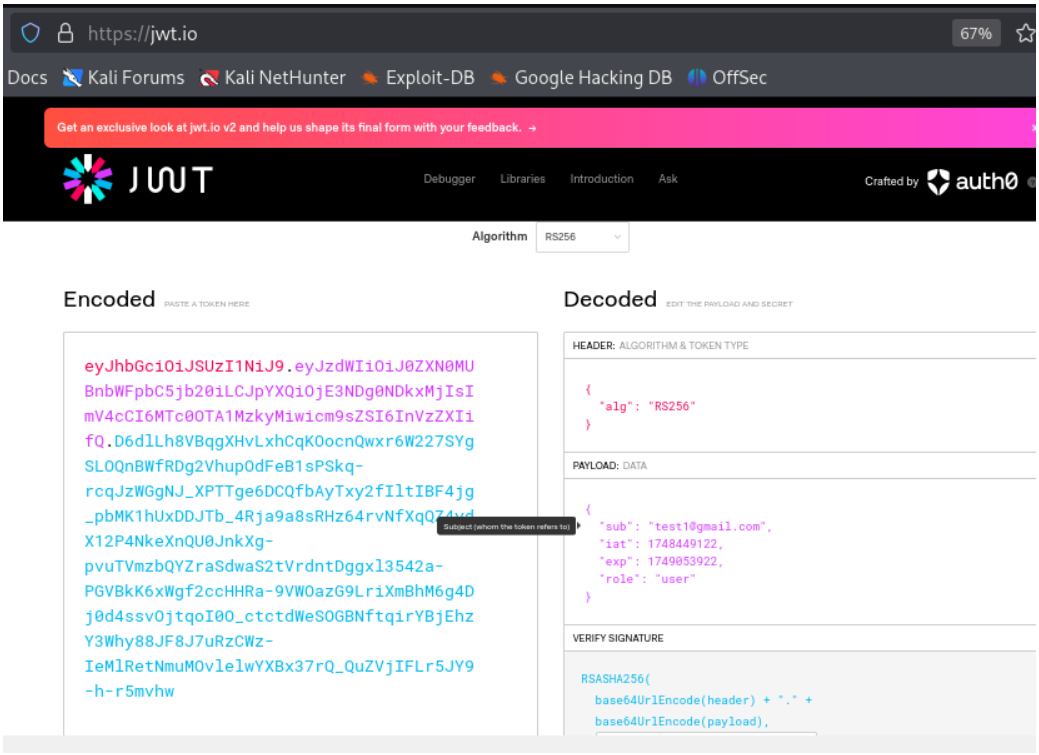


**Figura 38**

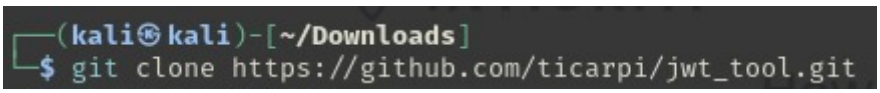
*Token decodificado*


## 4.5. GENERACIÓN DE TOKENS CON JWT TOOL

Para dar inicio al ataque propuesto hacia la API de la app crAPI, haremos uso de la herramienta “JWT-TOOL”, por lo que se detallara el proceso de su instalación y ejecución. En primer paso se clonará el repositorio de la herramienta por usar.

**Figura 39**

*Clonación de repositorio desde github*

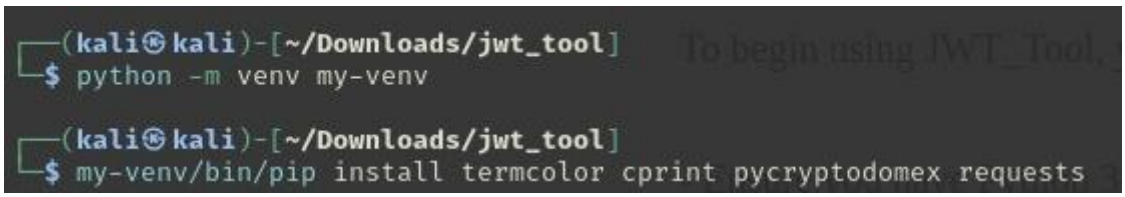



Luego de clonar el repositorio deseado, ingresamos a la carpeta “jwt_tool” y habilitaremos un ambiente virtual con los comandos “**Python -m venv**

**#ambiente#**" y dentro de este ambiente instalaremos las librerías correspondientes para que el programa jwt_tool.py pueda ejecutarse.

**Figura 40**

*Habilitación de ambiente virtual*




Tomar en cuenta la activación del ambiente virtual con el comando "source" y su ruta correspondiente según el nombre con el que hayamos creado el ambiente.

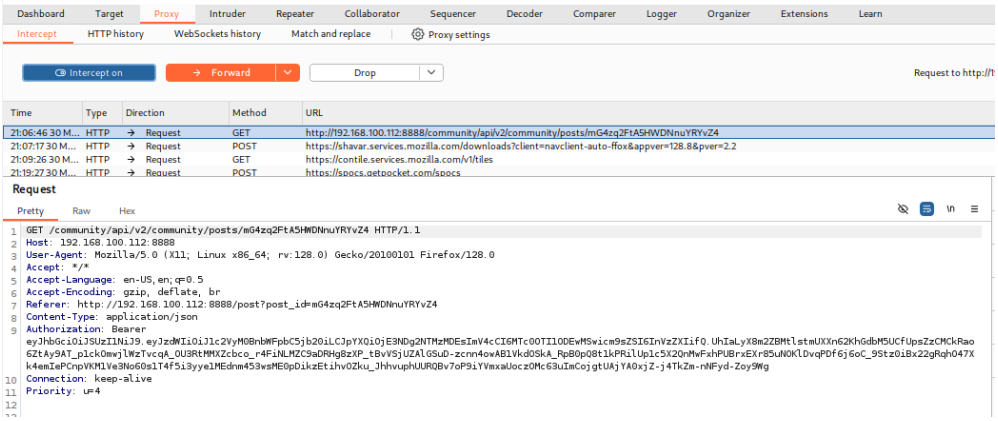
**Figura 41**

*Activación de ambiente y ejecución de programa JWT_TOOL*

Una vez nuestra herramienta esté operativa, procederemos a interactuar con la app crAPI interceptando sus request desde la herramienta Burpsuite.

**Figura 42**

*Request interceptado*




Interceptado el request, con el usuario ya iniciado, en el apartado de community, obtendremos el token Bearer, el cual procederemos a analizarlo en la app web jwt.io, visualizando su información correspondiente.

**Figura 43**

*Token de apartado community*


En esta ocasión podremos visualizar que el usuario con el que se ha iniciado sesión es el user3. Luego de esto, regresaremos a Burpsuite a analizar el request realizado a hacia la API del módulo community para trabajar así sobre el token generado por dicha API.

**Figura 44**

*API de sección Community*

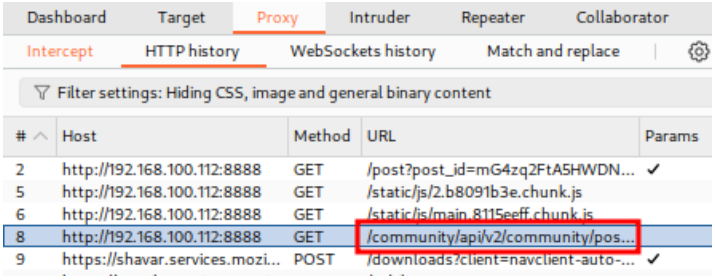
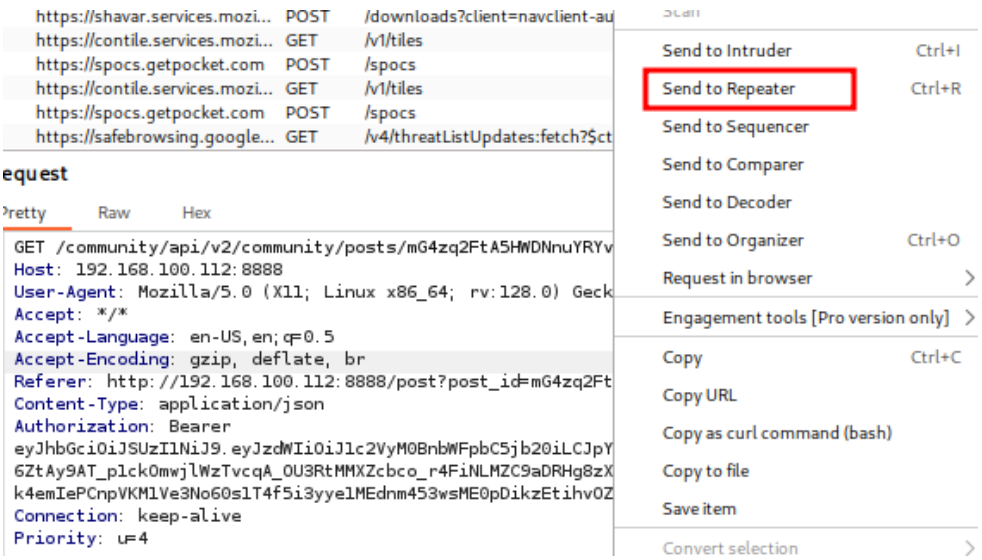


**Figura 45**

*Token original de sección Community*


**Figura 46**

*Envío de request a sección Repeater*




Se procederá a enviar el request obtenido al apartado de **"Repeater"** para interactuar con los datos generados por la herramienta JWT-Tool a continuación. Interactuando con JWT-Tool, con su opción de explotar vulnerabilidades (-X), procederemos a generar diferentes tokens con bearer autorizado que podrían ser aceptados por la app web crAPI al momento de validar dicho token. Ejecutaremos el comando**: "Python jwt_tool.py (**token**) -X a"**

**Figura 47**

*Ejecución de jwt_tool*

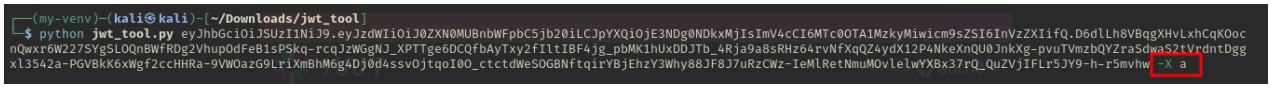
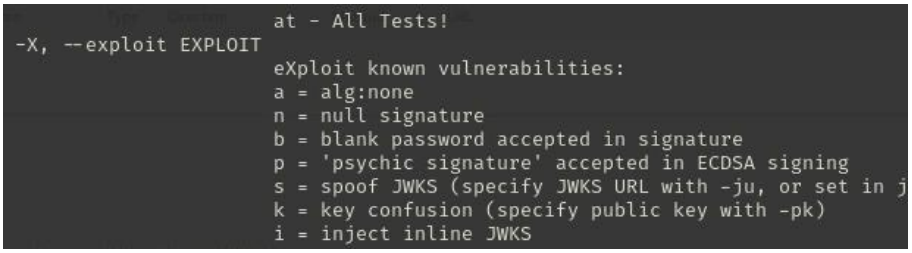



**Figura 48**

*Opciones por apartado de Exploit*




Como resultado visualizamos los diferentes tokens generados, que procederemos a introducir al request original a través de Burpsuite para validar si son aceptados con éxito por parte de la App web.

**Figura 49**

*Tokens genéricos*

Como previamente ya habíamos el enviado el request original a la sección de Repeater, modificaremos el token original por el generado por la herramienta jwt_tool verificando si obtenemos resultados exitosos.

**Figura 50**

*Token original*

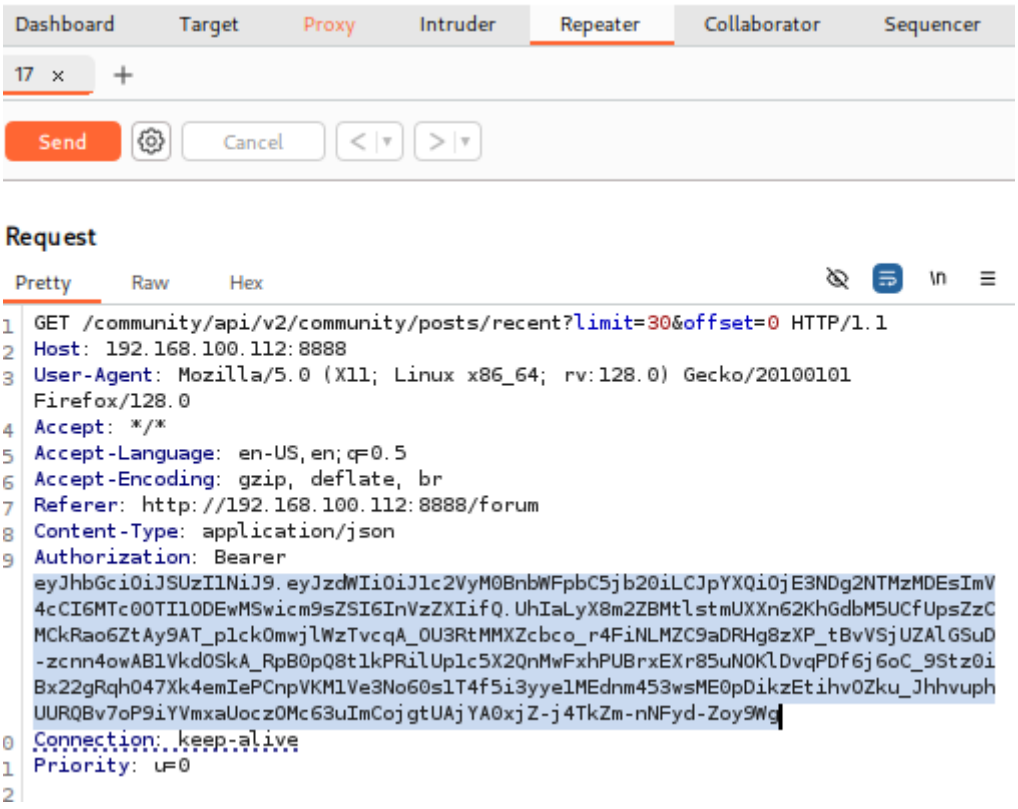


**Figura 51**

*Reemplazo por token genérico*

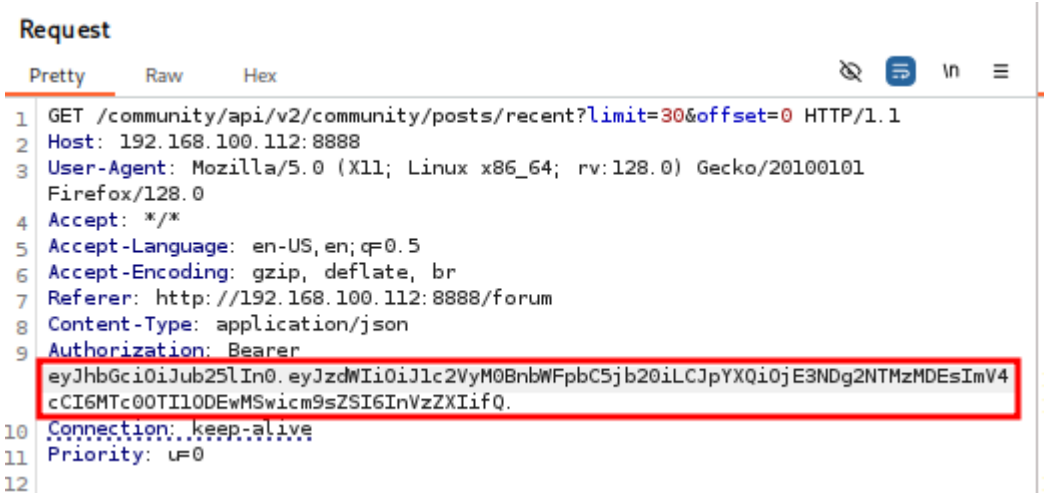
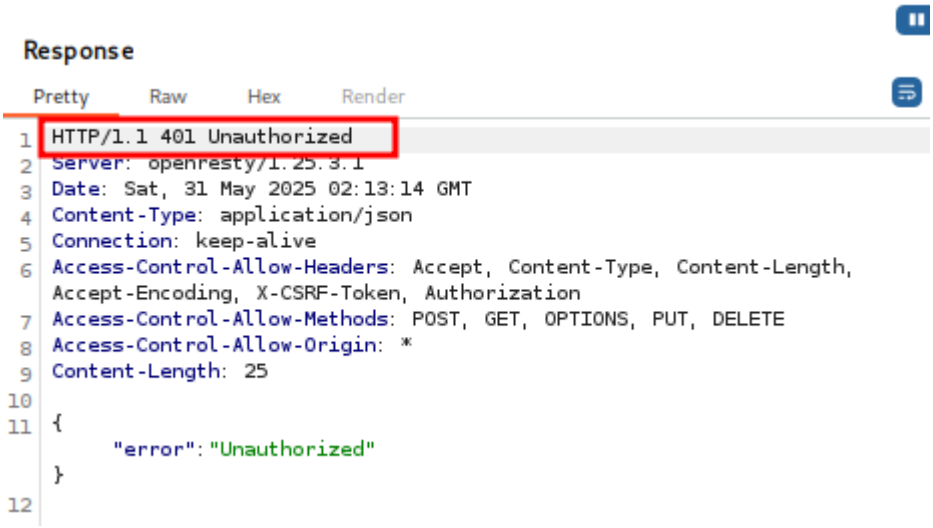



**Figura 52**

*Mensaje de error*

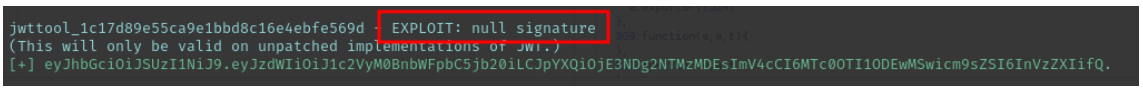



Luego de la modificación del token, se valida que esta app no es vulnerable a este tipo de ataque, y nos indica un error de no autorizado al envió del request con el token generado. En caso de que el token haya sido aceptado, se habría confirmado que la app no valida correctamente el token sin firma que le es

enviada. A modo de prueba realizaremos la misma actividad, pero con la opción de “**-X n**” para generar token con firma nula.

**Figura 53**

*Generación de token sin firma*




En la sección de Repeater intentaremos nuevamente utilizar el nuevo token generado y validaremos si es exitoso o no.

**Figura 54**

*Envío de request con token sin firma*


Se puede observar como respuesta el mensaje de no autorizado, por lo que la App web no es vulnerable a este tipo de ataques relacionado a tokens generados con la herramienta JWT_tool.

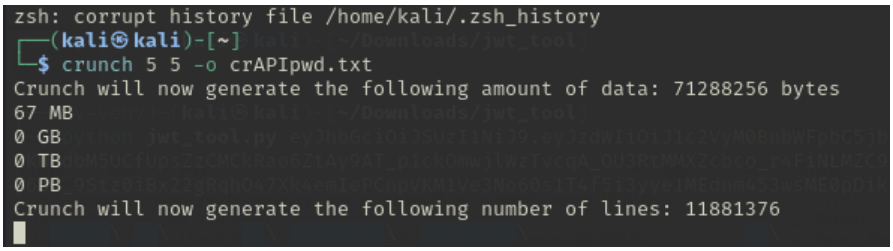
### 4.6. CRACKEO DE TOKEN

La meta de esta sección de la práctica es hallar la clave secreta encargada de firmar el token generado por la API, y siendo así poder alterar el contenido de un request, dicho contenido podría ser el usuario, correo, entre otros datos enviados en el request.

En primera instancia, haremos uso de un diccionario, el cual lo crearemos desde cero de la siguiente manera con la herramienta **crucnch**. Dicho diccionario lo generaremos por cadenas de hasta 5 dígitos.

**Figura 55**

*Creación de diccionario*




Para validar el diccionario creado podremos visualizarlo con el comando **nano** visualizando el contenido resultante, con el cual realizaremos un ataque de diccionario, tratando de validar así la clave que firma el token de la API de la aplicación.

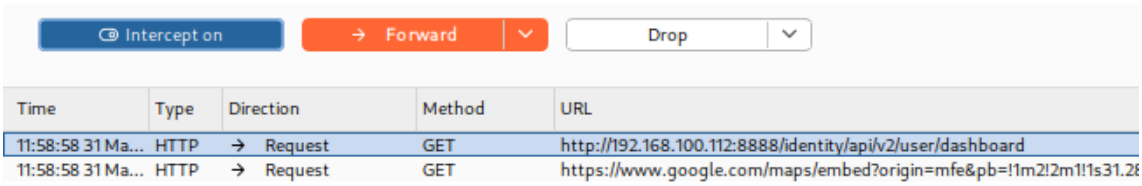
**Figura 56**

*Visualización de diccionario*


Con el diccionario creado, procederemos a logearnos en la app con un usuario creado, capturando el request de dashboard desde Brupsuite.

**Figura 57**

*Intercepción de API por dashboard*



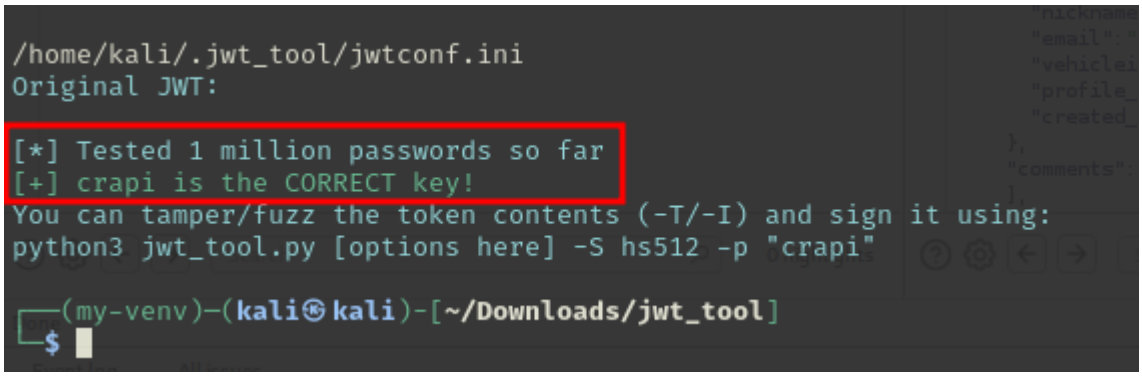
Ahora con ayuda de la herramienta jwt_tool utilizaremos su apartado de -C para el proceso de cracking en conjunto con el diccionario creado previamente. Ejecutaremos el comando**: "Python jwt_tool.py (**token**) -C -d (**diccionario**)".**

**Figura 58**

*Ejecución de jwt_tool por cracking*


**Figura 59**

*Hallazgo de clave de firma*




Al finalizar el proceso de crackeo, la herramienta nos indica que la contraseña del token es **"crapi"** por lo que sería posible alterar el token para cambiar información como usuario, correos entre otras.

**Figura 60**

*Estructura de token original*

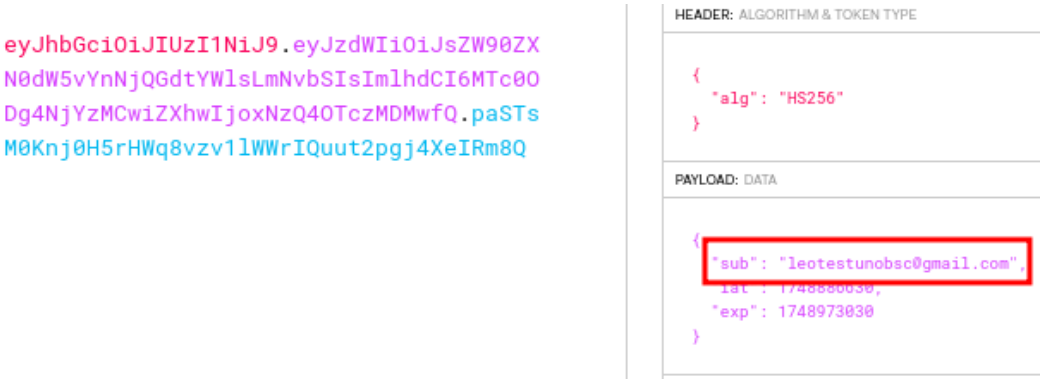



**Figura 61**

*Estructura de token alterado*


Ahora en el decodificador online ingresaremos manualmente un correo existente diferente del que estamos logeados, usaremos la contraseña “crapi” y se generara un nuevo token con el cual procederemos a enviar un request desde la sección de Repeater.

**Figura 62**

*Reemplazo de token alterado*

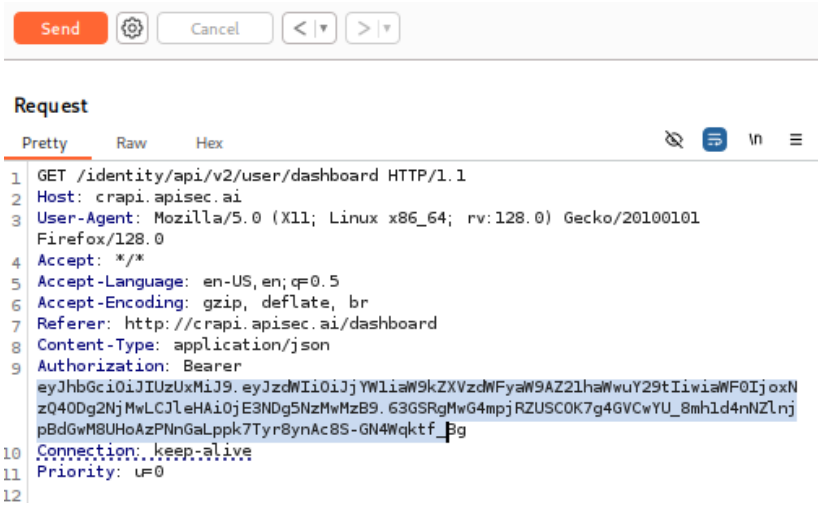



**Figura 63**

*Response con token alterado*


Validaremos en la sección de “Response” que el usuario y correo han sido cambiados mediante el token alterado que habremos generado con la clave

correspondiente. Esta vulnerabilidad permite realizar acciones y request con otros usuarios, correos, permitiendo así elevar permisos y suplantación de identidad.

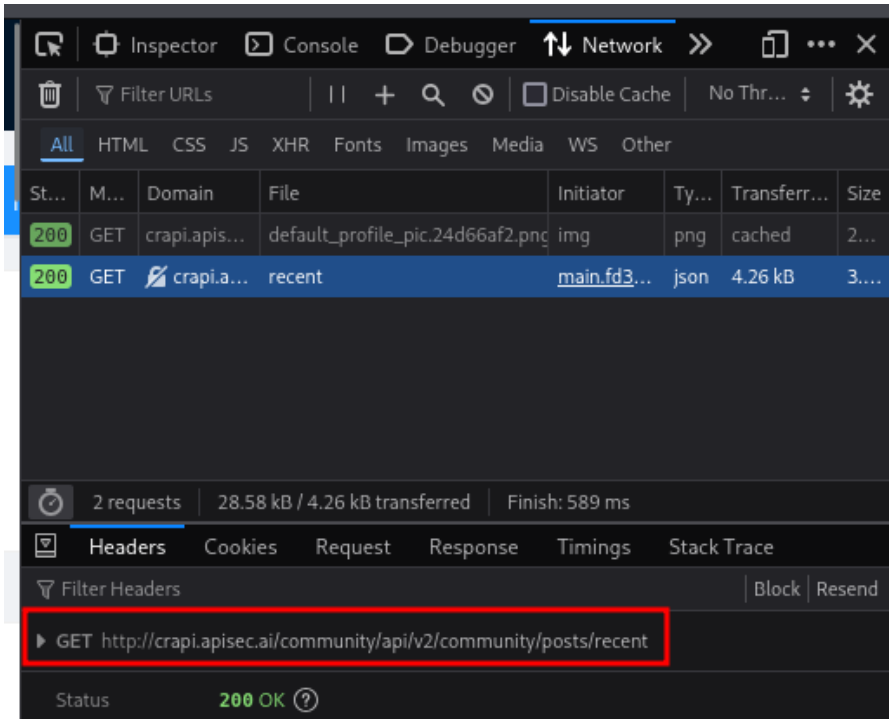
### 4.7. ATAQUE BOLA (BROKEN OBJECT LEVEL AUTHORIZATION)

Con el ataque que se tiene previsto realizar al servicio de crAPI se intentara acceder a información que debe ser solo accesible de cada usuario correspondiente. Por ejemplo, cada usuario registrado tiene la disponibilidad de registrar un vehículo único, este vehículo se asignará según un token y ID enviado por correo electrónico. Se prevé acceder a información de otros usuarios extrayendo el dato de sus vehículos.

Iniciaremos con el acceso al módulo de “Community” y con el inspector del navegador, en la sección de “Network” visualizaremos las request enviadas, visualizando la API sobre la cual trabajaremos.

**Figura 64**

*Intercepción de API desde navegador*




Luego en Postman pegaremos la url del api, enviando el método GET, obteniendo así información de aparatado de community visualizando información como nombres, ID, ID de autos, contenido y mail de los usuarios que han posteado en esta sección de la app.

**Figura 65**

*Envío de request a postman*

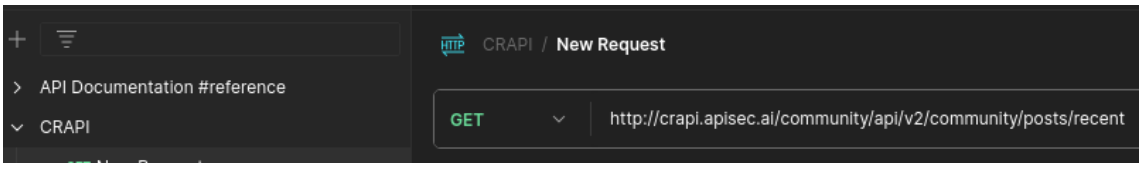


**Figura 66**

*Visualización de información del request*

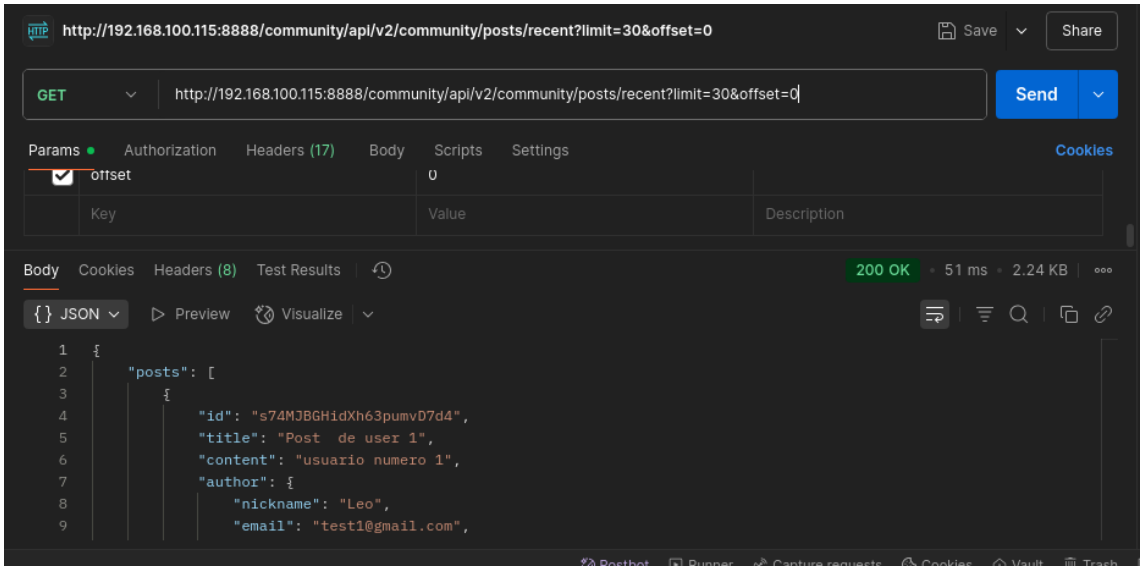
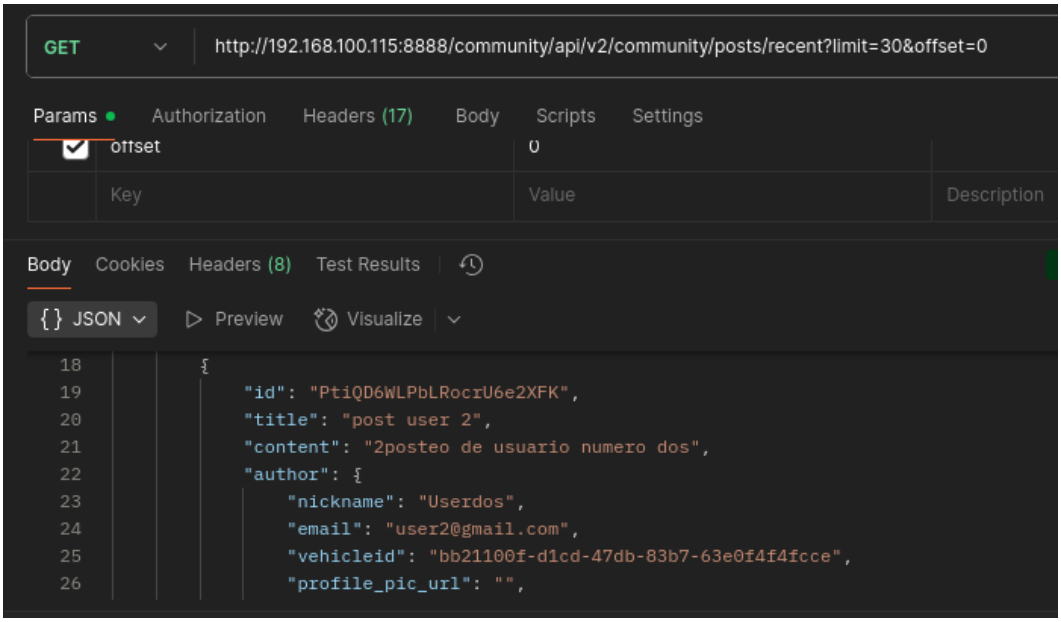



**Figura 67**

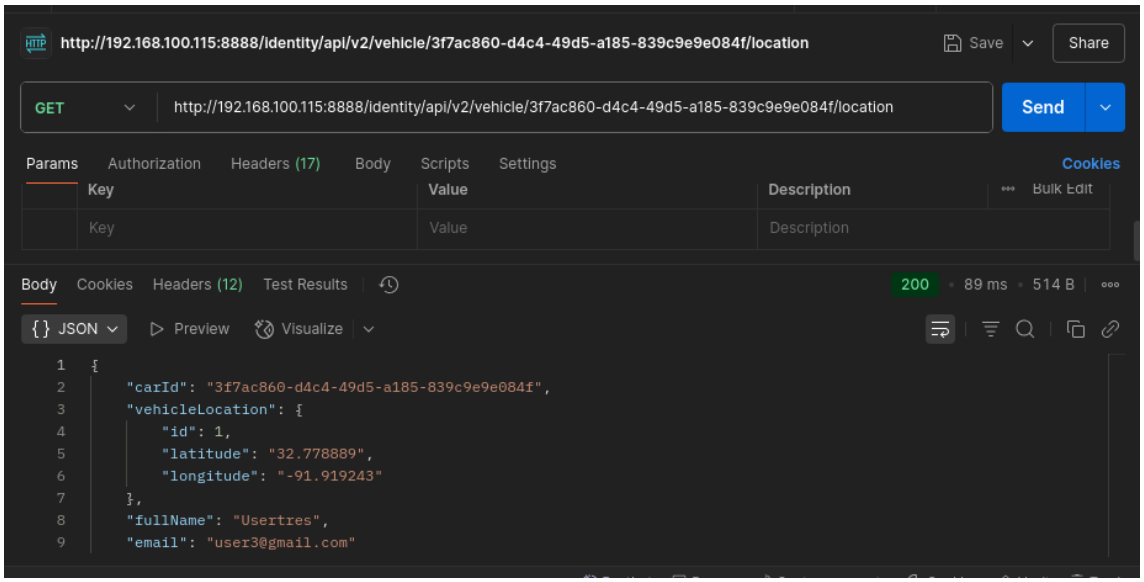
*Visualización de información del request 2*



Posterior a esto, visualizaremos el apartado de “Dashboard” para verificar información del vehículo del usuario tres obteniendo así la siguiente API recibida en Postman:

**Figura 68**

*Request de vehículo*




Como podemos visualizar, se detalla información del vehículo, ubicación, y correo del usuario al que pertenece. Con esta información obtenida, trataremos de acceder a información de otros usuarios con el ID del auto que hemos capturado desde el apartado de “Community “. Se reemplazará el “carId” tratando de obtener la información de los otros vehículos, ya que, si visualizamos el request de la API, el carID es parte de la URL, por lo que se espera que al reemplazar este ID por el de otro usuario, se acceda a información perteneciente a otro usuario.

Estando logeado como “usuario3” ingresaremos el “carId” del “usuario2” tratando de obtener dicha información.

**Figura 69**

*Información de vehicleid*

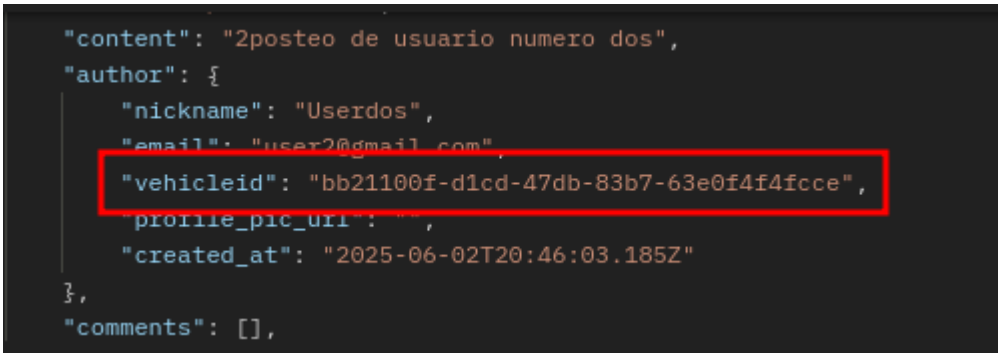
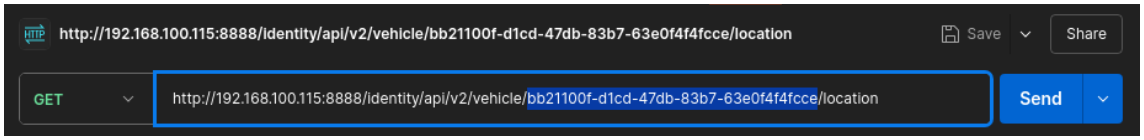



**Figura 70**

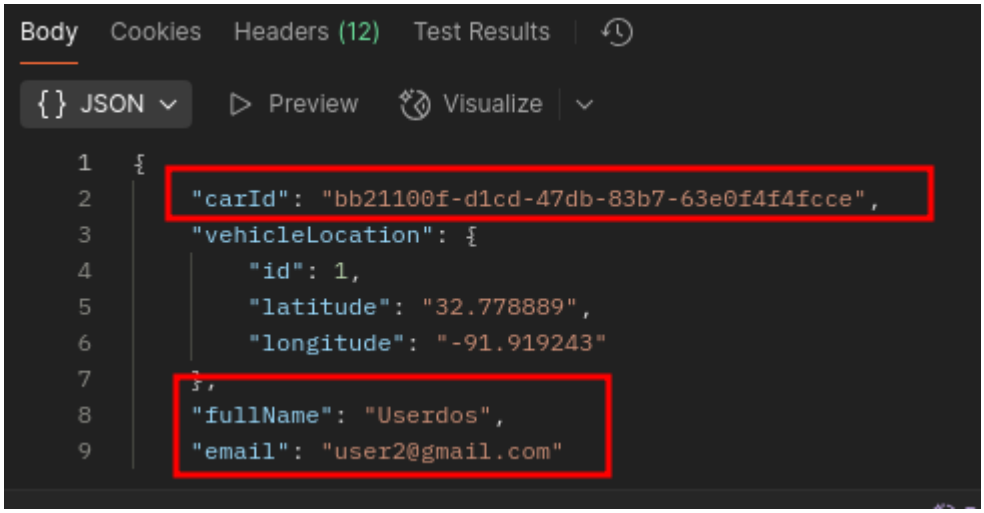
*Alteración en estructura de request*




Obteniendo así la información deseada del usuario 2, solo alterando la solicitud de la API con el detalle del **"carId",** validando así la posibilidad de extraer información de todos los usuarios únicamente alterando la estructura del request mediante la API y el ID del vehículo de cada usuario.

**Figura 71**

*Información de user2*




## 4.8. ATAQUE DE INYECCION SQL

Se realizará inyección de sentencias SQL en el apartado de "**Shop**" de la aplicación web crAPI con la finalidad de obtener información de artículos, usuarios o cupones disponibles por parte de la app. Al iniciar el proceso de "**Add coupons**" ingresaremos un cupón al azar, con la finalidad de interceptar este request por API a través del proxy de Burpsuite.

**Figura 72**

*Sección de "Shop" de la App web*

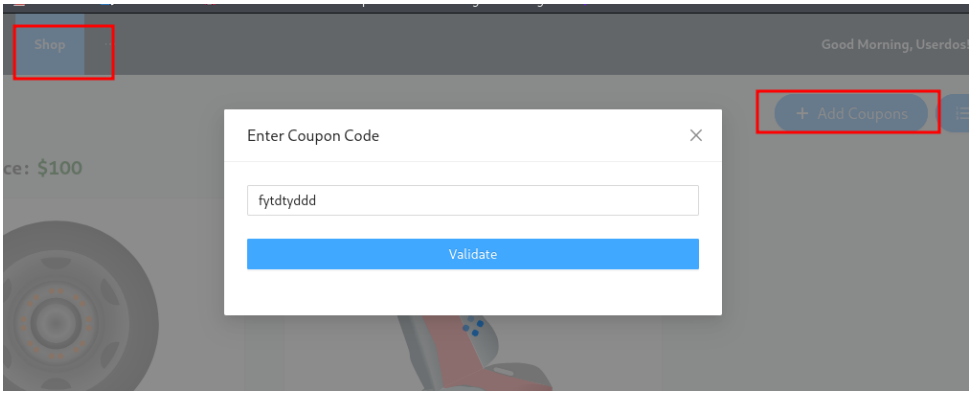
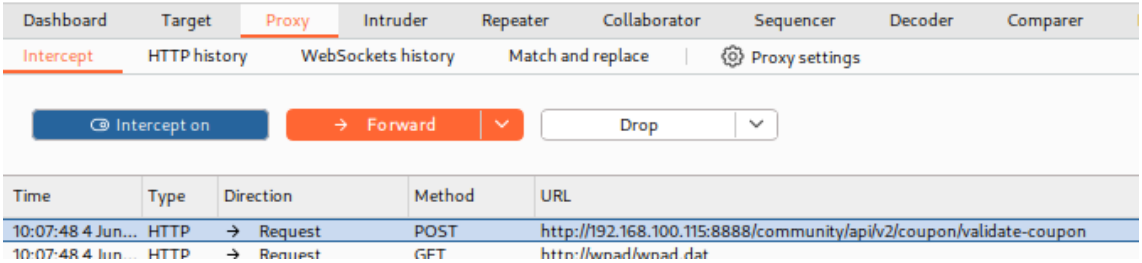



**Figura 73**

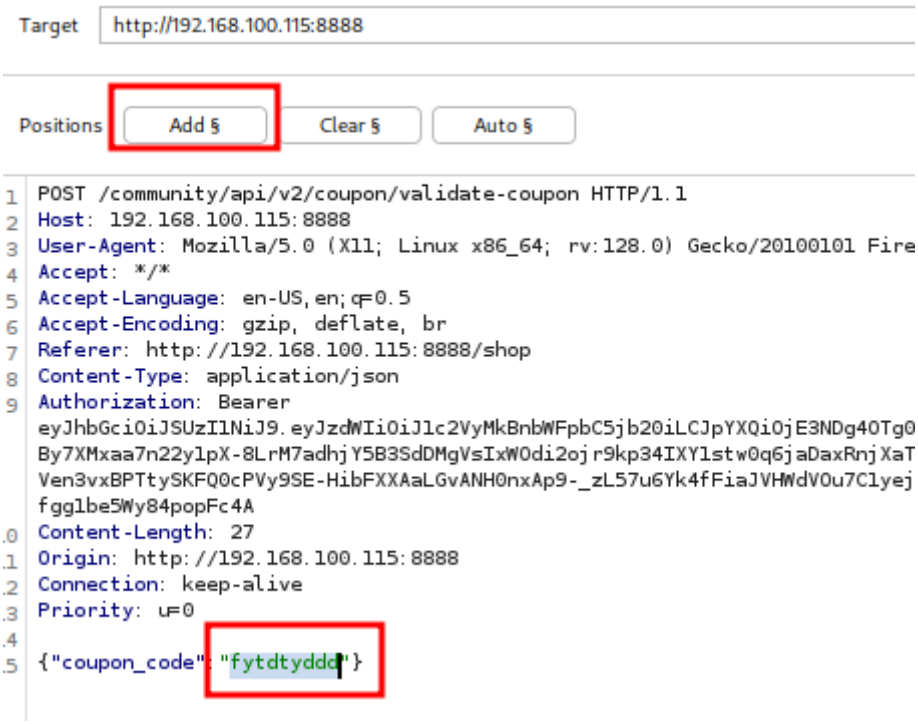
*Intercepción de request por validar cupón*




Con el request deseado ya interceptado, procederemos a enviarlo al apartado de "**Intruder**", donde cargaremos un archivo .txt con payloads por sentencias SQL, tratando así de vulnerar esta sección de la aplicación.

**Figura 74**

*Envío de request a "Intruder"*




Seleccionaremos la sección en la cual se cargará nuestro payload.

Realizaremos una búsqueda en internet por payloads que puedan vulnerar nuestra aplicación mediante sentencias SQL. Sin embargo, hay que tener en cuenta que la App trabaja con MongoDB, por lo que haremos uso de payloads por **noSQL** y **SQL.**

**Figura 75**

*Payload noSQL*

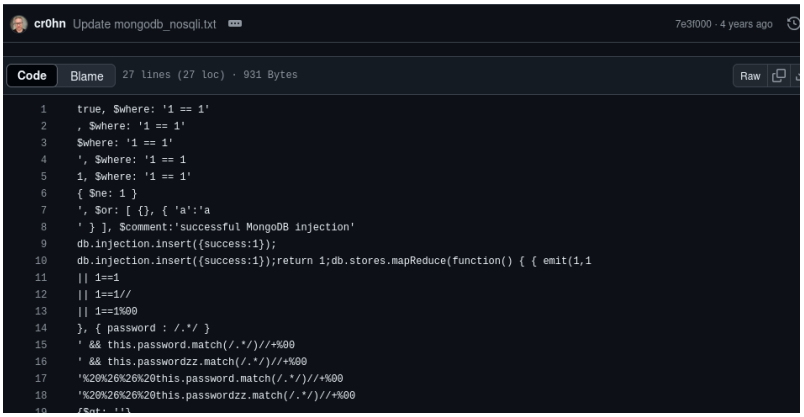
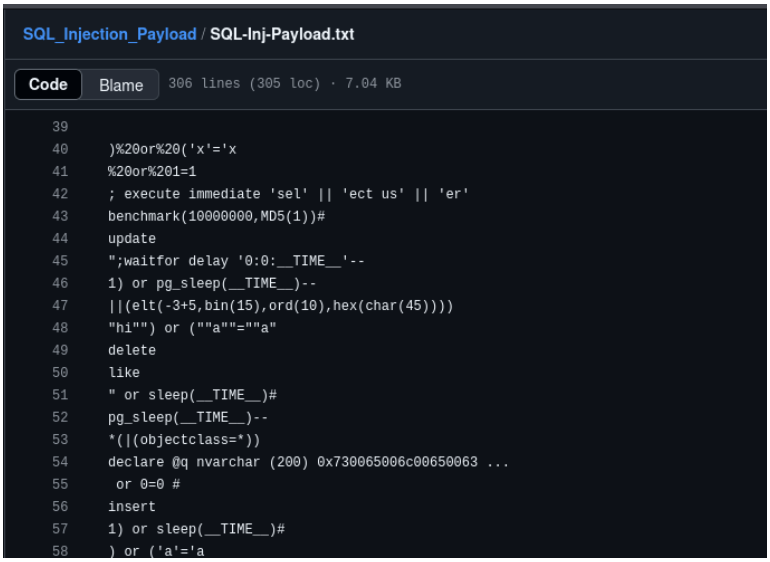



**Figura 76**

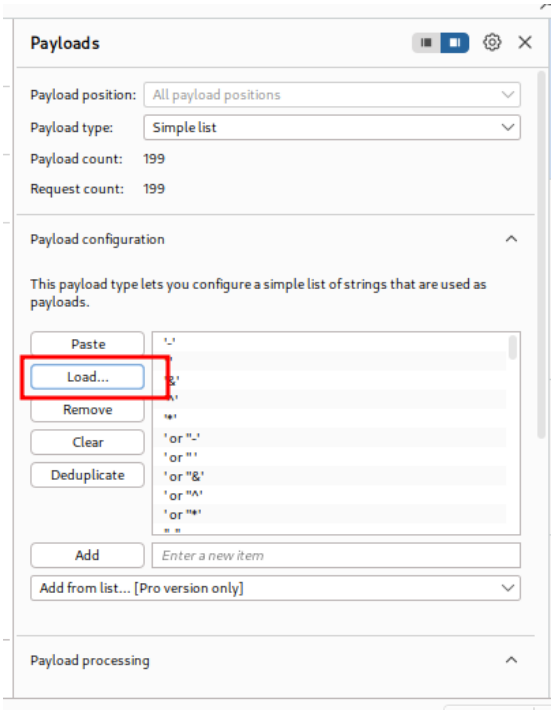
*Payload SQL*




Ya con los recursos necesarios, procederemos a iniciar el ataque planificado, esperando tener resultados exitosos. Cargaremos los archivos .txt de los payloads, tanto por SQL como noSQL.

**Figura 77**

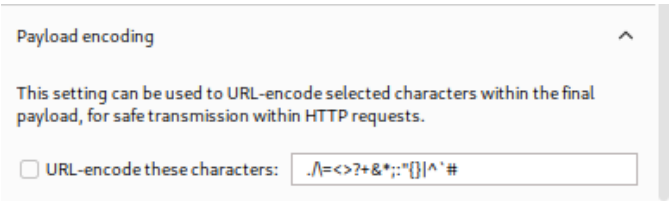
*Carga de Payload*




También deberemos desencasillar la opción de “Payload encoding” para que no se vea afectado la inserción de sentencias SQL durante el ataque.

**Figura 78**

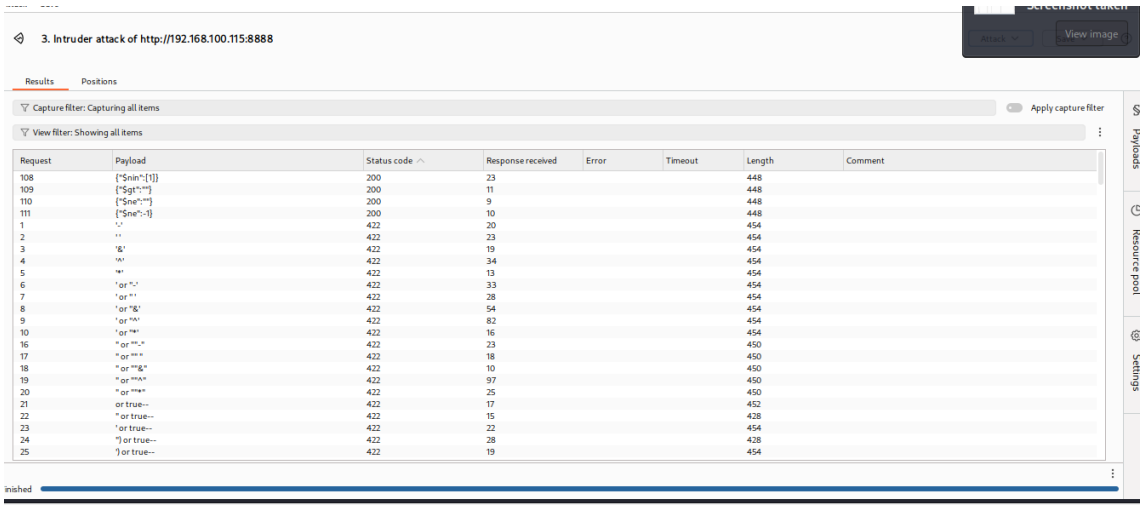
*Opción de codificar payload*




Una vez finalizado el ataque, procedemos a visualizar los códigos de estado, diferenciando así las sentencias que tuvieron resultados exitosos.

**Figura 79**

*Inicio de ataque*




Visualizando así las sentencias con código “200” que posiblemente serían las sentencias exitosas. Se procederá a revisar las respuestas de dichas sentencias para analizar los resultados.

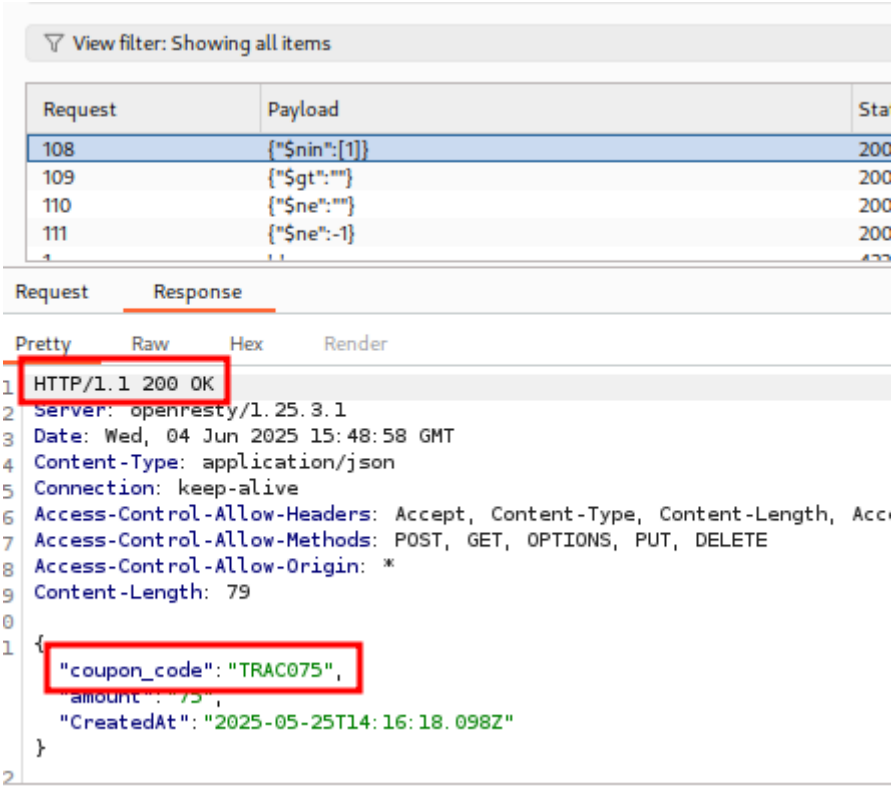
**Figura 80**

*Sentencias exitosas*

| Request | Payload | Status code |
|---|---|---|
| 108 | {"$nin":[1]} | 200 |
| 109 | {"$gt":""} | 200 |
| 110 | {"$ne":""} | 200 |
| 111 | {"$ne":-1} | 200 |

**Figura 81**

*Visualización de cupón*




Validando el código obtenido como cupón valido a través del ataque realizado por medio de la inyección SQL planteado. Adicional, investigando las sentencias que, si tuvieron efecto sobre la app, son sentencia **NoSQL** que tienen efecto directamente sobre MongoDB, la base de datos con la que trabaja la App crAPI.

## V. RESULTADOS

En base a los resultados obtenidos, podemos evidenciar las vulnerabilidades y nivel de riesgo que presentan el uso de APIs que se integran a programas web con mucha más frecuencia, sin tener en consideración la seguridad debida por parte de desarrolladores, ya que es mediante estas APIs que se envía y recibe información, se realizan transacciones, se ingresan usuarios y contraseñas, se inician y mantienen servicios, entre otros procesos.

Así mismo, es de suma importancia y crecimiento educativo realizar prácticas de ciberataques en un ambiente controlado o autorizado por parte de la entidad propietaria de la infraestructura sobre la cual se realizaría nuestro ataque, ya que es por medio de estas prácticas que se podrá comprender el comportamiento y funcionalidad de las APIs en integración con un programa backend o frontend, dando así apertura de saber cómo tomar acciones sobre las vulnerabilidades halladas.

Se recomienda directamente a los desarrolladores fortalecer la seguridad de las APIs al momento de implementar estas con sus programas, según su objetivo, fortaleciendo así las claves y validaciones JWT, validar autenticaciones de usuarios en cada request, implementar permisos según el rol del usuario, implementación de consultas parametrizadas, entre otros procesos seguros y estratégicos, que permitan al usuario final salvaguardar su información, dándole

confianza al momento de realizar transacciones, y manteniendo siempre presente la seguridad de la información.

## 5.1. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR CRACKEO DE JSON WEB TOKEN (JWT)

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define un formato seguro para la transmisión de claims entre partes en forma de objeto JSON firmado digitalmente. La estructura de un JWT consta de tres segmentos codificados en Base64 URL-safe, separados por puntos: Header, Payload y Signature.

En el desarrollo de aplicaciones web y servicios RESTful, la seguridad de las APIs es un pilar fundamental. Los JSON Web Tokens (JWT) se han convertido en una solución popular para la autenticación y manejo de sesiones sin estado(stateless) en estos entornos. Un JWT es esencialmente un token firmado digitalmente que transporta información del usuario, permitiendo a múltiples servidores validar la identidad sin necesidad de almacenar sesiones guardadas.

Si un JWT es crackeado (comprometido o falsificado), un atacante podría obtener acceso no autorizado a recursos protegidos, suplantar identidades o escalar privilegios dentro del sistema. (Karthikeyan Nagaraj, 2023).

La importancia de asegurar los JWT radica en que un fallo de seguridad puede traducirse en el compromiso de un API. De hecho, OWASP destaca la autenticación vulnerada en su top10 de riesgos de seguridad para APIS, donde los JWT mal implementados podrían generar varios escenarios como robo de token, manipulación de firmas, secuestro de cuentas, escalamiento de privilegios y filtración de datos. (Nitzan Namer, 2023).

**Ataques más comunes contra JWT**

**Manipulación de algoritmo:** Uno de los ataques más conocidos contra JWT, debido a que aprovecha fallos en la validación del algoritmo de firma. En donde, el ataque consiste en engañar al servidor para que no valide la firma o se pueda usar una clave incorrecta. Existen dos variantes principales: Algoritmo "None" y Confusión entre HMAC y RSA.

Algoritmo "none", hace referencia a que, en algunas bibliotecas antiguas o código mal configurado, un atacante puede modificar el encabezado de un JWT (alg = none), para que elimine la firma y el servidor pueda aceptar el token como valida sin verificar su integridad. Esto permite al atacante inclusive tomar el token de otro usuario, elevar privilegios (cambiando el rol a admin) y así obtener acceso no autorizado. (Akamai, 2023).

Confusión entre HMAC y RSA, Ocurre cuando el servidor acepta tokens tanto simétricos como asimétricos, pero no verifica que el algoritmo coincida con

lo esperado. El atacante se encarga de conseguir la clave publica de la aplicación. Posterior a esto crea un JWT malicioso cambiando el encabezado, logrando así firmar un token usando una clave publica como si fuese una clave secreta. Así el atacante obtiene un token aparentemente valido para el servidor. (OWASP, 2025).

**Fuerza Bruta de la clave secreta:** En tokens firmados con algoritmos simétricos (HMAC), la fortaleza del JWT depende totalmente de la calidad de la clave secreta. Si la clave es débil (de longitud corta, débil), un atacante puede realizar un ataque de fuerza bruta offline. Para lograr esto, el atacante, deberá interceptar un JWT válido y posterior a esto probar con diccionarios de claves hasta poder encontrar cuál de las firmas coincide. Herramientas como Jhon the Ripper o Hashcat permiten automatizar esta búsqueda rápidamente. (OWASP, 2025).

La potencia de cómputo actual ha reducido drásticamente el tiempo necesario para lograr crackear passwords comunes: una clave de 8 caracteres que hace pocos años habría tomado hasta 10 días en romperse, hoy se la podría romper en menos de 5 días. En un escenario de ataque offline, el atacante no genera tráfico que pueda delatarlo, ya que simplemente toma el token y lo trabaja fuera de la vista de la víctima, lo cual dificulta detectar este tipo de ataques. (OWASP, 2025).

**Ataques de replay:** Se produce cuando un JWT válido es capturado y reutilizado maliciosamente por un atacante. A diferencia de sistemas de sesión

tradicional, el JWT una vez emitido, mientras no expire, cualquier presentación del token con una firma valida será aceptada. Si un atacante intercepta el token de un usuario, ya sea mediante el ataque *Man in the middle* en un canal inseguro, o más comúnmente extraído vía XSS. Podría repetir peticiones suplantando al usuario legitimo. Dado que el servidor no almacena estado o, no hay forma nativa de distinguir una petición legítima de un replay usando el mismo JWT. El atacante podría duplicar solicitudes o acceder a información sin necesidad de conocer credenciales. Este ataque en si no rompe la firma del JWT, sino que explota la falta de unicidad de uso.

**Abuso del parámetro Key ID (Kid) y otros encabezados:** El encabezado o header de JWT puede incluir campos opcionales como Key ID, que indica al servidor que clave usar para verificar la firma. Implementaciones inseguras pueden utilizar directamente este valor para buscar la clave en el store o sistema de archivos, permitiendo así el paso a ataques de inyección. Estos ataques dependen de fallos específicos en el código, pero demuestran la superficie adicional que presentan los JWT más allá del algoritmo (OWASP, 2025).

**Prevención de ataques**

**Elegir algoritmos seguros y fijar la política de validación:** Nunca permitir algoritmos no seguros. Es obligatorio configurar la librería JWT para no aceptar tokens con alg=none bajo ninguna circunstancia. De igual manera, si la aplicación utiliza un tipo de algoritmo específico (p. ej., RS256), debe rechazar tokens que no

coincidan con ese algoritmo esperado en el header Se recomiendan algoritmos asimétricos modernos; OWASP sugiere ECDSA sobre RSA por su solidez criptográfica y eficiencia, aunque RSA sigue siendo seguro si se maneja correctamente. (Akamai, 2025).

**Gestionar apropiadamente las claves de firma:** Las claves privadas en algoritmos asimétricos deben mantenerse en servidores seguros, preferiblemente en módulos de seguridad de hardware (HSM) o almacenes tipo Vault, minimizando la posibilidad de filtración. Las claves simétricas igualmente deben tratarse como secretos críticos. No se deben reutilizar la misma clave en múltiples servicios o dominios. Cada aplicación o microservicio debería tener su propio secreto para firmar JWT. Así se evita que un token emitido en un contexto sea válido en otro. Documentos de Akamai recalcan que usar la misma clave privada en diferentes aplicaciones puede llevar a robo de cuentas entre ellas. Por tanto, segmentar el uso de claves es fundamental. (Akamai, 2025).

**Configurar tiempos de expiración adecuados**: Un principio esencial es limitar la vida útil de cada JWT mediante el claim exp (Expiration Time). Tokens de corta duración reducen la ventana de tiempo en que un token robado o crackeado es útil para el atacante. Según las recomendaciones, sesiones web podrían tener tokens con expiración de 15 minutos a 1 hora, combinados con mecanismos de refresh seguros si se requiere mantener al usuario autenticado. En aplicaciones sensibles, incluso menor.

**Validaciones exhaustivas en el backend por cada uso de JWT:** El servidor no debe confiar jamás en un JWT sin validarlo completamente. Cada vez que llega un token, se deben realizar al menos las siguientes comprobaciones:

Verificar la firma digital con la clave correcta y el algoritmo esperado (como ya se indicó, no usar funciones de decodificación sin verificación). Muchas vulnerabilidades surgen cuando los desarrolladores usan métodos que decodifican el JWT, pero omiten involuntariamente la verificación de firma. Esto debe evitarse configurando adecuadamente la biblioteca de JWT para siempre validar.

Comprobar los claims de control: que no esté expirado (exp > ahora), que si tiene nbf ya sea válido, que el iss (issuer) corresponda a quien debería emitirlo (evita tokens de terceros), que el aud (audience) coincida con nuestra aplicación si se usa ese campo, etc. Un check estricto de las fechas de emisión/expiración previene tokens reusados fuera de tiempo.

Asegurarse de la integridad del formato: rechazar tokens mal formados, ignorar campos inesperados en el header (para evitar ataques de confusión usando encabezados que apunten a claves externas, a menos que explícitamente se soporte y valide su uso).

**Monitorear y limitar intentos en endpoints de autenticación:** Ataques como fuerza bruta de contraseñas, podrían involucrar probar firmas contra un

endpoint de verificación si existiera (menos común, ya que normalmente se haría offline). Sin embargo, mantener alertas sobre actividades anómalas en las APIs (múltiples intentos fallidos de autenticación con JWT, tokens con campos sospechosos) puede ayudar a detectar un ataque en tránsito. Implementar rate limiting en endpoints sensibles y registros detallados puede ser útil para descubrir patrones de ataque.

**Medidas correctivas ante compromisos de JWT**

A pesar de las medidas preventivas, existe la posibilidad de que algún JWT sea comprometido (robo de token no expirado, o una vulnerabilidad 0-day explotada). En estos escenarios, es clave reaccionar rápidamente para limitar daños. Estas son las principales medidas correctivas y de prevención:

**Revocación de tokens (Denylist/Blocklist):** Dado que por diseño JWT no tiene un mecanismo interno de invalidación temprana, una contramedida imprescindible tras detectar un token comprometido es anular su vigencia antes de que expire.

**Cambio de claves de firma en caso de brecha:** Si se sospecha que la clave secreta o privada fue comprometida, la medida inmediata es cambiar dicha clave y redistribuirla de forma segura (en el caso de HMAC) o actualizar los certificados (en caso de RSA/ECDSA). Al cambiar la clave, todos los JWT existentes firmados con la anterior se invalidarán (al no poder verificarse con la

clave nueva). Esto desconectará a todos los usuarios activos, por lo que debe usarse en caso de emergencia o idealmente planificado en mantenimientos.

**Detección de anomalías (Anomaly Detection):** Implementar monitoreos que identifiquen usos sospechosos de JWT puede considerarse una medida de prevención y reactiva a la vez. Por ejemplo, si se observa que el mismo token JWT se utiliza casi simultáneamente desde dos ubicaciones geográficas muy distantes, o desde direcciones IP distintas, podría ser señal de token robado.

**Investigación y auditoría post-incident**: Como parte fundamental de las acciones correctivas tras un ciberataque, resulta esencial realizar una auditoría detallada de los registros de actividad (logs). Este análisis permite identificar los recursos que fueron accedidos mediante el JWT vulnerado, establecer la duración del uso indebido del token y rastrear las direcciones IP involucradas. Esta información es clave para evaluar posibles exposiciones de datos sensibles y, en consecuencia, determinar si es necesario aplicar protocolos de notificación a las partes afectadas o emprender acciones legales conforme a las normativas vigentes.

## 5.2. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR ATAQUE BOLA

La Autorización de Nivel de Objeto Roto es una vulnerabilidad de seguridad cuando una aplicación o interfaz de programación de aplicaciones (API), proporciona accesos a objetos basados en un rol de usuario, pero no verifica si el

usuario está autorizado para acceder a esos recursos. Esta vulnerabilidad permite a los atacantes saltarse las seguridades, accediendo a los datos o ejecutar acciones no autorizadas (Berenguel Gómez & Sánchez, 2024).

Esta vulnerabilidad esta categorizada como CWE-639 que significa omisión de autorización mediante clave controlada por el usuario, por lo que esta vulnerabilidad aparece cuando un determinado usuario requiere acceder a un recurso (archivos, ficheros, datos del usuario, etc.) y los mecanismos de seguridad pueden ser omitidos, lo que puede llegar a causar que se filtre información sensible, divulgación, modificación y dar acceso a funcionalidades restringidas (Berenguel Gómez & Sánchez, 2024).

**Escenarios de ataque BOLA**

Un escenario donde comúnmente se repiten estos tipos de ataques son las plataformas de comercio electrónico, el atacante puede identificar los endpoints de las APIs que se ocupan como fuente de datos, identificar un patrón, modificar la URL de la solicitud y obtener los datos del comercio (OWASP, 2023).

Otro escenario común es en las plataformas que gestionan documentos en línea para múltiples usuarios, cuando se crea una solicitud de eliminación de un documento de un usuario en concreto, la solicitud suele llevar el secuencial o el número de documento a eliminar por lo que si no se controla el usuario al que

pertenece el archivo y acceso al mismo, cualquier usuario puede eliminar los documentos de otro (OWASP, 2023).

Esta vulnerabilidad puede manifestarse de diferentes maneras:

**Solicitud de información de un usuario**: Una página web mediante la URL "https://mysite.com/profile?user_id=**123456**" carga la información del usuario actual, sin embargo, sin las medidas de seguridad adecuadas esta solicitud puede ser modificada cambiando el ID del usuario y cargando la información de otros usuarios.

**Tabla 1**

*Ejemplo de Endpoint para obtener la información de un usuario por ID*

| API | Resultado |
|---|---|
| GET /api/user_profile/2222222222 | Acceso Correcto |
| GET /api/user_profile/2222222223 | Datos de Otro usuario, BOLA |

*Nota*. Esta tabla muestra cómo se construye una solicitud para obtener los datos de un usuario por el ID.

**Manipulación de peticiones REST**: Similar al caso anterior se puede modificar la petición de un método GET "/apirest/shop/cart/4", en caso de no tener un control se puede listar la información de otro usuario.

**Acceso a ficheros estáticos mediante URL:** En el caso de que no se realice un control sobre lo que puede ver y acceder un usuario un atacante podría

modificar el nombre el archivo y acceder los ficheros que pueden contener información sensible, un ejemplo de URL vulnerable sería “https://mysite/files/1234.pdf” donde el nombre del documento es un secuencial.

**Como los atacantes explotan esta vulnerabilidad**

**Identificando la vulnerabilidad:** El primer paso en cualquier ataque es identificar la vulnerabilidad. Los atacantes generalmente observan cómo la solicitud forma su URL o API. Si la aplicación usa referencias directas a objetos internos, es una señal clara de que la aplicación puede ser vulnerable a través de BOLA.

**Manipulando la referencia:** Cuando el atacante identifica la posible vulnerabilidad, el siguiente paso es manipular la referencia al objeto. Puede ser tan simple como cambiar el número de número de URL, pero puede ser muy complicado, dependiendo de la estructura de la URL y la lógica de la aplicación. Si la aplicación no ha autorizado correctamente la aplicación, el atacante puede acceder a los datos o tomar medidas en nombre de otro usuario. Esto puede conducir a una violación de datos, robo de identidad y varios otros problemas de seguridad.

**Acceso no autorizado:** El paso final luego de la manipulación exitosa de la referencia es el acceso no autorizado. Si la información del usuario está expuesta

el atacante puede utilizar esta misma información para poder realizar otros tipos de ataque entre algunos: Phishing, Robo de Identidad, Ingeniería Social, entre otros.

**Prevención y mitigación de la vulnerabilidad**

La prevención de este tipo de ataques contempla varios niveles, entre los más importantes está el desarrollo de software bajo buenas prácticas, esto quiere decir que se debe considerar la implementación de controles de acceso a nivel de objeto por código, la validación de los accesos y permisos de un usuario sobre la información y los recursos.

Prevención mediante autenticación y autorización, tener un buen esquema de autenticación facilita la gestión de los permisos de un usuario, esto se puede conseguir con la implementación de estándares como JWT y OAuth, además se puede potenciar mediante una correcta implementación de patrones de control de acceso a usuarios que entre los más comunes están RBAC, ABAC y DAC.

El nivel de pruebas es uno de los aspectos más importantes para poder mitigar las vulnerabilidades que se pueden haber pasado por alto en niveles anteriores, por lo que es recomendable realizar pruebas y auditorias de los desarrollos mediante herramientas como BurpSuit, Postman, OWASP ZAP que permiten interceptar peticiones y detectar accesos indebidos. Además, se debe

implementar pruebas automatizadas e inspección del código fuente, los endpoints de API pueden detectarse mediante revisión de código o pruebas de penetración.

Cuando está implementada de manera correcta la autorización a objectos, solo los usuarios con la respectiva autorización pueden acceder a dichos recursos. Esto se logra mediante identificadores únicos, que puedan identificar y corroborar que sea el usuario respectivo, por ejemplo, ID del usuario, rol o id del objeto al que se necesita acceder. De esta manera se controla de manera eficaz que no se acceda a los recursos cuyos accesos no se encuentra autorizados.

Es importante la implementación de políticas específicas para endpoints cuando se encuentra alguna vulnerabilidad, estas políticas deben considerar la protección de los servicios expuestos públicamente y de ser el caso implementar un middleware de seguridad en frameworks actuales y con alta fiabilidad.

## 5.3. PREVENCIÓN Y MEDIDAS CORRECTIVAS POR ATAQUES DE INYECCIÓN SQL EN APIS

En la actualidad, las interfaces de programación de aplicaciones (APIs) se han convertido en elementos fundamentales del desarrollo de software, facilitando la interoperabilidad entre sistemas. Sin embargo, esta

interconectividad también genera nuevos vectores de ataque que pueden ser aprovechados por actores maliciosos.

Una de las amenazas más comunes es la inyección SQL (SQL Injection), una técnica de explotación que permite manipular consultas SQL al enviar entradas maliciosas a través de formularios o solicitudes web. Cuando los datos no se validan correctamente, los atacantes pueden acceder, modificar o eliminar información crítica (OWASP, 2023).

Además de comprometer la integridad y confidencialidad de los datos, estos ataques pueden utilizarse como punto de partida para otras amenazas, como el escalado de privilegios, la denegación de servicios o incluso la persistencia en sistemas vulnerables. Por lo tanto, comprender cómo ocurren estos ataques, cómo prevenirlos y cómo reaccionar ante ellos es esencial para proteger infraestructuras críticas.

**Tabla 2**

*Ciclo de ataque típico de una inyección SQL en APIs*.

| **Etapa** | **Descripción** |
|---|---|
| Entrada maliciosa | El atacante inserta código SQL en parámetros de entrada. |
| Falta de validación | El backend no filtra o sanitiza adecuadamente la entrada. |
| Ejecución de consulta | La base de datos interpreta el código como parte de la lógica de consulta. |
| Acceso o daño | El atacante accede, modifica o borra información crítica. |

**Naturaleza del ataque: ¿Cómo se produce la inyección SQL en APIs?**

La vulnerabilidad de inyección SQL surge cuando los datos de entrada del usuario se integran directamente en las sentencias SQL sin ser validados ni parametrizados. En APIs, esto ocurre típicamente mediante solicitudes HTTP como GET o POST, que transportan los datos al backend de la aplicación. Si el backend ejecuta una consulta construida dinámicamente con estas entradas sin filtrarlas adecuadamente, se corre el riesgo de que un atacante incluya instrucciones SQL maliciosas. Esta falla no solo compromete la confidencialidad, sino también la integridad y disponibilidad de los datos (Halfond, Viegas & Orso, 2006).

En APIs RESTful, donde la lógica se basa en recursos accesibles mediante URL y cuerpo de solicitudes, los vectores de ataque incluyen encabezados personalizados, parámetros en la URL, y cuerpos JSON con estructuras manipuladas. Una falta de validación en estos puntos puede ser aprovechada por atacantes para insertar comandos SQL arbitrarios.

### Factores que propician la vulnerabilidad

**Tabla 3**

*Factores que propician la vulnerabilidad*.

| Factor de riesgo | Descripción |
|---|---|
| Concatenación directa de cadenas SQL | Uso inseguro de variables en consultas sin parametrización. |
| Falta de validación de entradas | Entradas del usuario sin filtros o restricciones. |
| Uso incorrecto de ORM | ORM mal configurados que permiten inyecciones en consultas dinámicas. |
| Configuraciones débiles de seguridad | Ajustes predeterminados que facilitan el acceso no autorizado. |
| Ausencia de control de acceso basado en rol | Usuarios con privilegios excesivos. |

### Estrategias preventivas contra la inyección SQL en APIs

**Tabla 4**

*Estrategias preventivas contra la inyección SQL en APIs*.

| Estrategia | Beneficio clave |
|---|---|
| Consultas parametrizadas | Separan los datos de la lógica, evitando ejecución de código malicioso. |
| Validación de entradas | Asegura que solo datos esperados ingresen al sistema. |
| Gestión de privilegios | Limita el impacto en caso de vulneración. |
| Uso de WAF | Detección y bloqueo de patrones de ataque comunes. |
| SDLC seguro | Minimiza riesgos desde etapas tempranas del desarrollo. |

**Medidas correctivas ante una inyección SQL detectada**

Ante la detección de un ataque de inyección SQL, es crucial activar un protocolo de respuesta inmediata que contemple:

- Identificación del origen del ataque y análisis forense.
- Revisión del código para ubicar y eliminar la vulnerabilidad.
- Implementación de controles preventivos como parametrización o sanitización.
- Actualización de componentes vulnerables (librerías, frameworks).
- Notificación a las partes afectadas y, si aplica, a los entes reguladores.
- Revalidación de la seguridad mediante nuevas pruebas de penetración.

## Herramientas de seguridad útiles

**Tabla 5**

*Herramientas de seguridad útiles*.

| Herramienta | Funcionalidad principal |
|---|---|
| OWASP ZAP | Escaneo dinámico de vulnerabilidades en aplicaciones web y APIs. |
| SQLMap | Detección automatizada de vulnerabilidades de inyección SQL. |
| Burp Suite | Auditoría de seguridad de aplicaciones web mediante pruebas manuales y automáticas. |
| Postman y Newman | Validación de APIs y pruebas automatizadas mediante scripts. |
| SonarQube | Análisis estático del código para hallar fallas de seguridad. |

### Prácticas recomendadas en el diseño seguro de APIs

- Autenticación fuerte basada en estándares modernos (OAuth 2.0, JWT).
- Comunicación segura mediante protocolos como HTTPS con certificados válidos.
- Registro detallado de eventos (logging) y monitoreo de actividad sospechosa.
- Implementación de controles de acceso detallados y revisiones periódicas de roles.
- Uso de entornos separados para pruebas, desarrollo y producción.

- Integración de pruebas de seguridad en cada etapa del ciclo de vida del software.
- Uso de límites de tasa (rate limiting) y control de concurrencia para evitar ataques de fuerza bruta o automatizados.

**Impacto de incidentes y casos reales**

Casos emblemáticos como el de TalkTalk en el Reino Unido, donde se comprometieron datos personales por no aplicar prácticas básicas de seguridad, sirven como lecciones fundamentales. Heartland Payment Systems también sufrió consecuencias económicas y legales al ser víctima de una inyección SQL. Estos eventos demuestran cómo una falla técnica puede transformarse en una crisis reputacional y financiera de gran escala.

En Latinoamérica, también se han reportado ataques similares a plataformas de e-commerce y entidades bancarias, revelando la importancia de realizar auditorías periódicas de seguridad incluso en sistemas aparentemente estables.

**Cultura organizacional y formación en seguridad**

- Las organizaciones deben adoptar un enfoque holístico hacia la ciberseguridad. Esto implica:

- Capacitar continuamente a los desarrolladores en técnicas de codificación segura.
- Fomentar una mentalidad de seguridad desde la concepción del producto.
- Implementar prácticas DevSecOps que integren la seguridad en el flujo de trabajo.
- Promover la colaboración entre equipos de desarrollo, QA y seguridad.
- Incentivar el reporte interno de vulnerabilidades mediante canales éticos y seguros.

**Normativas y cumplimiento**

Los marcos regulatorios como el GDPR y la CCPA exigen a las empresas proteger los datos de los usuarios mediante controles adecuados. Las infracciones derivadas de inyecciones SQL no solo comprometen la privacidad, sino que también pueden acarrear sanciones legales severas. Por lo tanto, cumplir con estas normativas no es solo una obligación legal, sino una responsabilidad ética.

Asimismo, existen normas como ISO/IEC 27001 y NIST que orientan sobre cómo gestionar la seguridad de la información, proponiendo controles técnicos y administrativos aplicables a entornos de APIs.

### Automatización en la detección de vulnerabilidades

La automatización de las pruebas de seguridad se ha vuelto esencial en entornos ágiles. Los pipelines CI/CD pueden incorporar herramientas como:

**Tabla 6**

*Automatización en la detección de vulnerabilidades*.

| Técnica/Prueba | Descripción |
|---|---|
| SAST (Análisis Estático) | Evalúa el código fuente para detectar errores antes de la ejecución. |
| DAST (Análisis Dinámico) | Identifica vulnerabilidades durante la ejecución de la aplicación. |
| IAST | Combina lo mejor de SAST y DAST para un análisis más profundo. |
| Dependabot / Snyk | Verifican versiones y vulnerabilidades en dependencias de software. |

## VI. ANEXOS

Repositorio de descarga por máquinas virtuales implementadas:

https://drive.google.com/drive/folders/1cp-VuWFCG36RMJvMS8JZ3ltfzUHU48xe?usp=sharing

(Enlace disponible hasta septiembre 18 de 2025)

## VII. REFERENCIAS

Adeel Ehsan,Mohammed Ahmad. (26 April 2022) RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions

https://www.mdpi.com/2076-3417/12/9/4369

AKAMAI (s/f) *¿Cuáles son los riesgos de seguridad de las API?*

https://www.akamai.com/es/glossary/what-are-api-security-risks

AWS (s/f) ¿Qué es una API?

https://aws.amazon.com/es/what-is/api

AWS (s/f) ¿Qué es una API RESTful?

https://aws.amazon.com/es/what-is/restful-api

Beas, J. F. J. V. (2020). Diseño e implementación de un API para la gestión de servicios de computo en la nube del CUValles. https://riudg.udg.mx/visor/pdfjs/viewer.jsp?in=j&pdf=20.500.12104/81994/1/MCUVALLES10019FT.pdf

Chrystal R. China, Michael Goodwin. (s/f*) ¿Qué es la seguridad de las API?*

https://www.ibm.com/es-es/think/topics/api-security

Cibersafety. (s/f) *¿Qué es Burp Suite? Guía Completa de la Herramienta de Ciberseguridad para Proteger Aplicaciones Web Vulnerables*

https://cibersafety.com/que-es-burp-suite-herramienta-ciberseguridad/

CLOUDFLARE (s/f*) ¿Qué es la seguridad de la API?*

https://www.cloudflare.com/es-es/learning/security/api/what-is-api-security/

Edward Amoroso. (01/06/2020) Comprender las Ciber amenazas a las APIs. Centro de Respuesta a incidentes se Seguridad Informática.

https://www.csirt-epn.edu.ec/como-tener/90-ciberamenaza-apis

Erinjeri, J. (2022). REST API for the Weary Beginner. Proceedings of the SouthEast SAS Users Group (SESUG) 2022, 179. https://sesug.org/proceedings/sesug_2022_final_papers/Learning_SAS_I/SESUG2022_Paper_179_Final_PDF.pdf

FORMADORES IT (s/f) ¿Qué es Postman? ¿Cuáles son sus principales ventajas?

https://formadoresit.es/que-es-postman-cuales-son-sus-principales-ventajas

Golmohammadi, A., Zhang, M., & Arcuri, A. (2024). Testing RESTful APIs: A Survey. *ACM Transactions on Software Engineering and Methodology*, *33*(1), 1–41.

https://doi.org/10.1145/3617175

Halfond, W. G. J., Viegas, J., & Orso, A. (2006). A Classification of SQL Injection Attacks and Countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering*, 10–17.

Informatica Documentation. (s/f) *JSON web token authentication*

https://docs.informatica.com/integration-cloud/cloud-api-manager/current-version/api-manager-guide/authentication-and-authorization/json-web-token-authentication.html

Jessica Barker. (2025) Hackeados: Los secretos detrás de los ciberataques. Granica.

Karthikeyan Nagaraj. (Nov. 2023). JWT attack vulnerabilities in Web App Penetration Testing | 2023, Infosec Matrix

Neumann, A., Laranjeiro, N., & Bernardino, J. (2021). An Analysis of Public REST Web Service APIs. En IEEE Transactions on Services Computing (Vol. 14, pp. 957–970).

https://doi.org/10.1109/TSC.2018.2847344

Nitzan Namer. (junio 2023) Análisis de las amenazas de autenticación de usuarios comprometida para JSON Web Tokens. Akamai Security Research.

https://www.akamai.com/es/blog/security-research/owasp-authentication-threats-for-json-web-token

Optiv. (2018). Getting started with Postman for API security testing: Part 2.

https://optiv.com/insights/discover/blog/getting-started-postman-api-security-testing-part-2

Optiv Blog. (2019). Better API penetration testing with Postman – Part 1.

Security Boulevard.

https://doi.org/10.1109/TSC.2018.2847344

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/

OWASP. (2023) OWASP Top 10 API Riesgos de seguridad

https://owasp.org/API-Security/editions/2023/en/0x11-t10/

OWASP Foundation. (2022) Testing JSON Web Tokens (WSTG-SESS-10). OWASP Web Security Testing Guide

https://owasp.org/www-project-web-security-testing-guide/latest/4-eb_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens

PortSwigger. (2025). Using Burp to test a REST API.

https://portswigger.net/support/using-burp-to-test-a-rest-api portswigger.net

Postman Blog. (2023). What are HTTP methods?

https://blog.postman.com/what-are-http-methods/

Postman Docs. (2025, abril 30). Monitor health and performance of your APIs in Postman.

https://learning.postman.com/docs/monitoring-your-api/intro-monitors/

Pynt. (2024). API testing with Burp Suite: A practical guide.

https://www.pynt.io/learning-hub/burp-suite-guides/api-testing-with-burp-suite-a-practical-guide/

QALIFIED BUILDING QUALITY. (09/08/2023) ¿Cómo utilizar Postman para las pruebas de API?

https://qalified.com/es/blog/postman-para-api-testing/

RadView. (2024). Advanced guide to API performance testing.

https://radview.com/blog/blog-api-performance-testing-advanced-guide/

*REDHAT (s/f) ¿Qué es una API REST?*

*https://www.redhat.com/es/topics/api/what-is-a-rest-api*

Scarfone, K., Souppaya, M., Cody, A., & Orebaugh, A. (2020). Technical Guide to Information Security Testing and Assessment (NIST Special Publication 800-115). National Institute of Standards and Technology.

https://doi.org/10.6028/NIST.SP.800-115

Shaaban, A. (2024, junio 21). HTTP methods explained: Understanding GET, POST, PUT, and DELETE in RESTful API design. LinkedIn.

https://www.linkedin.com/pulse/http-methods-explained-understanding-get-post-put-delete-shaaban-nqyyf/