

# Desentrañando la Seguridad en APIs: Guía para Pentesters

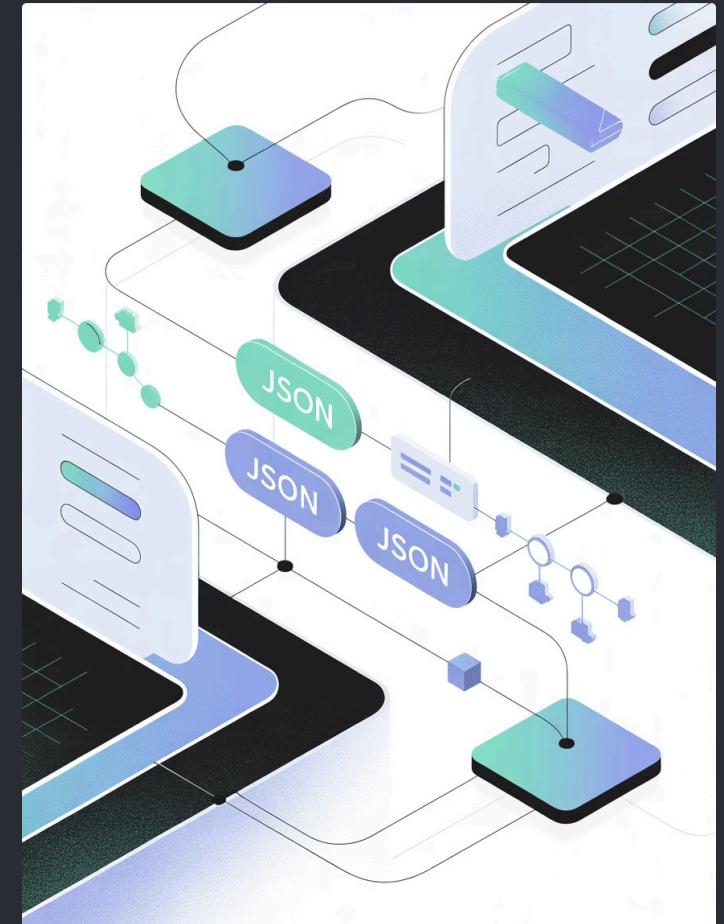
Una inmersión profunda en la arquitectura, reconocimiento y técnicas de ataque para asegurar tus APIs.



# ¿Qué es una API Técnicamente?

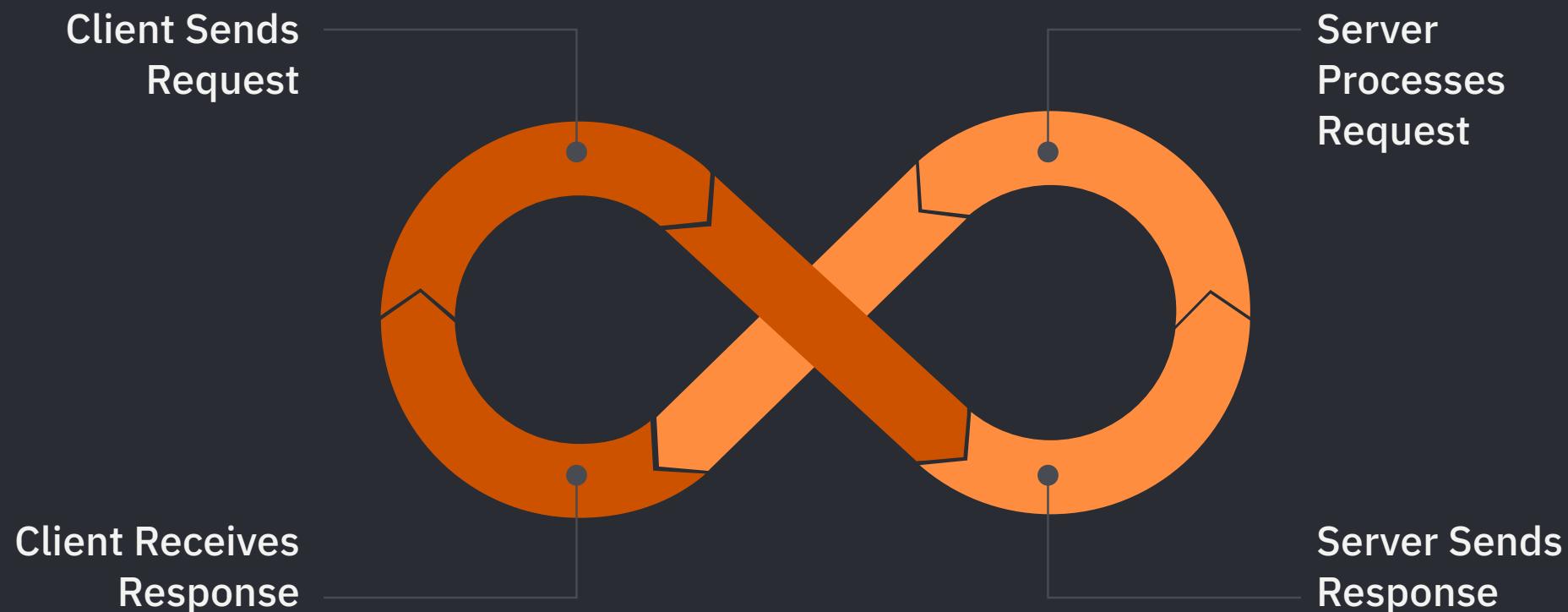
Una **Interfaz de Programación de Aplicaciones (API)** actúa como un puente digital, permitiendo la comunicación estandarizada entre diferentes componentes de software, típicamente un cliente (navegador, aplicación móvil) y un servidor.

Su función es abstraer la complejidad del sistema subyacente, exponiendo solo las operaciones necesarias para interactuar con él de manera segura y eficiente.



- El formato JSON (JavaScript Object Notation) se ha consolidado como el estándar de facto para el intercambio de datos en APIs RESTful debido a su ligereza y facilidad de lectura.

# Ciclo Request/Response: El Diálogo entre Cliente y Servidor



Cada interacción con una API sigue un ciclo predecible. La petición del cliente contiene elementos cruciales que definen la operación deseada y los datos enviados, mientras que la respuesta del servidor indica el resultado de esa operación.

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>Headers</b> Metadatos sobre la petición o la respuesta.	<b>Path</b> El recurso específico al que se accede (ej. /users/123).	<b>Query Params</b> Filtros o criterios adicionales (ej. ?name=john).	<b>Body</b> Los datos principales enviados (POST, PUT, PATCH).

# Verbos HTTP: Definiendo la Intención



## GET

Recupera datos del servidor. Nunca debe modificar el estado del recurso.



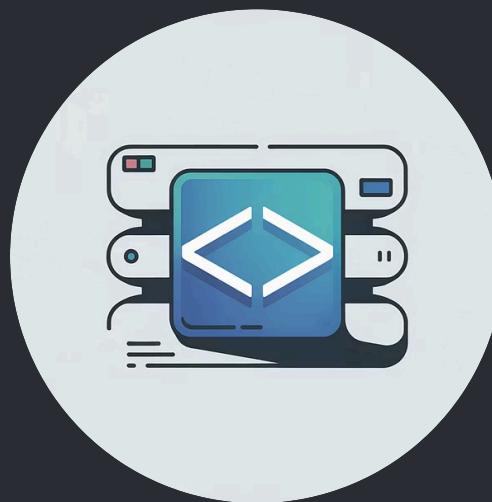
## POST

Envía datos para crear un nuevo recurso.



## PUT

Actualiza completamente un recurso existente o lo crea si no existe.



## PATCH

Actualiza parcialmente un recurso existente.



## DELETE

Elimina un recurso específico del servidor.

Un uso incorrecto del verbo HTTP puede generar vulnerabilidades o comportamientos inesperados, especialmente si un GET modifica datos.

# Autenticación en APIs: Verificando la Identidad



La autenticación es el proceso de verificar la identidad de un cliente que intenta acceder a una API.

Existen varios mecanismos, cada uno con sus propias implicaciones de seguridad:

- **Bearer Tokens (JWT):** Un token firmado que prueba la identidad y autorizaciones. Enviado en el header `Authorization: Bearer [token]`.
- **API Keys:** Claves únicas para identificar el origen de la petición. A menudo se envían como header o parámetro de consulta.

□ La correcta gestión y el secreto de estos credenciales son fundamentales para prevenir accesos no autorizados.

# Superficie de Ataque de una API

## Frontend (Interfaz)

- Interacciones visibles por el usuario.
- Validaciones a nivel de navegador.
- Errores de configuración en la interfaz.

## Backend (Puntos de Entrada de Datos)

- Endpoints expuestos directamente.
- Validaciones a nivel de servidor.
- Manejo de datos sensibles.

Es crucial entender que atacar una API va más allá de la interfaz de usuario. Los **puntos de entrada de datos** directos al backend son la principal superficie de ataque para un pentester de APIs.

# Reconocimiento Pasivo: Desvelando Endpoints Ocultos

El reconocimiento pasivo busca información sobre la API sin interactuar directamente con ella, minimizando la huella. Es como un detective que busca pistas en el entorno.



## Archivos Públicos

Buscar en directorios web, sitemaps o archivos robots.txt.



## Repositorios

Explorar GitHub, GitLab o Bitbucket en busca de código fuente expuesto.



## Scripts de Cliente (.js)

Analizar archivos JavaScript en busca de llamadas a la API y rutas ocultas.



# Fuzzing de Endpoints: Descubriendo Rutas Ocultas

El fuzzing consiste en enviar un gran volumen de peticiones malformadas o inesperadas a una API para provocar errores o descubrir comportamientos no documentados.

## Estrategias Clave:

- **Enumeración de Directorios:** Intentar rutas comunes como `/v1/`, `/api/`, `/admin/`, `/dev/`.
- **Extrapolación de Versiones:** Si se encuentra `/v2/`, probar `/v1/` o `/v3/`.
- **Archivos de Respaldo:** Buscar `.bak`, `.old`, `.zip` de endpoints.



- Esta técnica es una forma de fuerza bruta dirigida, indispensable para cartografiar la superficie de ataque completa de una API.

# Análisis de Documentación: Leyendo el "Mapa" de la Aplicación



La documentación de una API es una fuente invaluable de información para un pentester. Proporciona una visión clara de cómo se supone que debe funcionar la API.

- **Swagger/OpenAPI:** Estándares para describir APIs RESTful, revelando endpoints, métodos, parámetros, modelos de datos y esquemas de autenticación.
- **RAML:** Otro lenguaje para describir APIs, ofreciendo detalles similares.

Busca en la documentación:

- Endpoints sensibles o desprotegidos.
- Parámetros con validaciones débiles.
- Versiones antiguas que puedan tener vulnerabilidades conocidas.

# Burp Suite para APIs: Interceptando el Flujo



Burp Suite es una herramienta indispensable para la auditoría de seguridad de aplicaciones web y APIs.

## Configuración del Proxy:

- Permite interceptar todo el tráfico HTTP/S entre el cliente y el servidor.
- Esencial para ver peticiones que el navegador no muestra directamente.

## Historial y Manipulación:

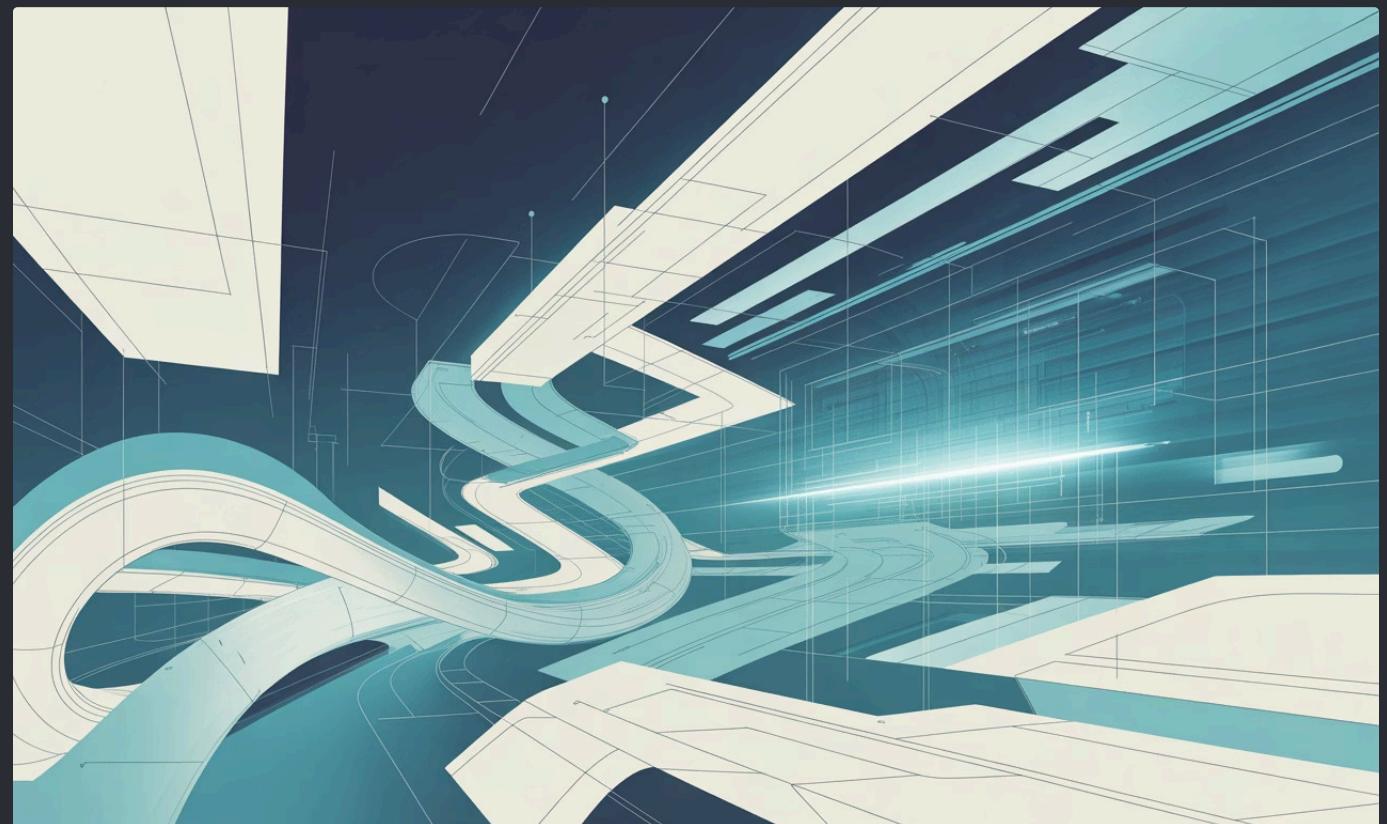
- El historial del proxy registra cada petición y respuesta, facilitando el análisis.
- Funcionalidades como Repeater e Intruder son clave para modificar y reenviar peticiones con payloads específicos.

Dominar Burp Suite es fundamental para el reconocimiento activo y la explotación de vulnerabilidades en APIs.

# Exposición de Datos y Gestión de Activos

## Vulnerabilidad: Information Exposure

Cuando una API revela información sensible como detalles técnicos, rutas internas o versiones de software en sus respuestas. Esta información puede ser utilizada por atacantes para perfilar la infraestructura y encontrar posibles puntos débiles.





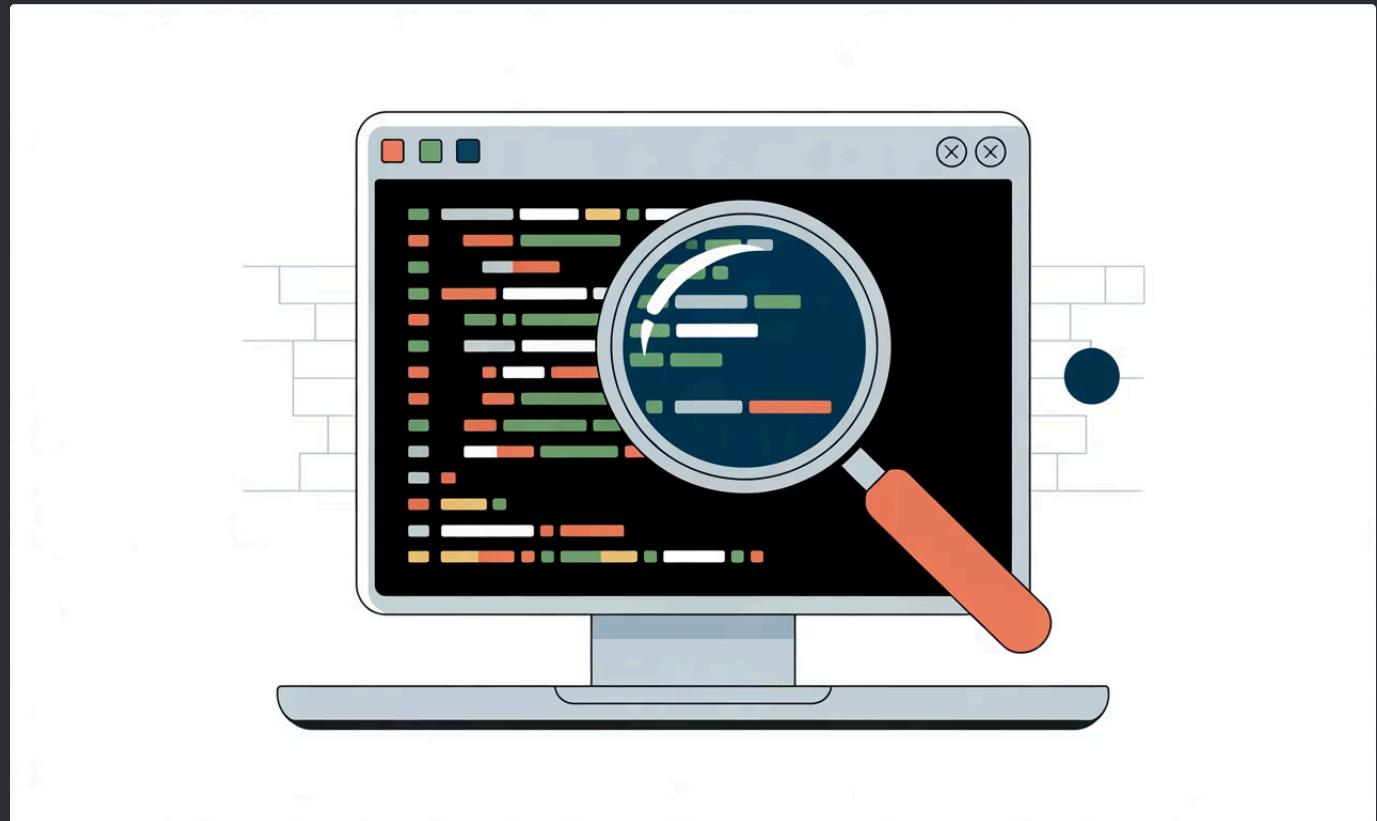
## Improper Assets Management

El peligro de las APIs "zombis": versiones obsoletas o de prueba (ej. `/beta/`, `/v1/`) que carecen de los parches de seguridad adecuados. Estas APIs, a menudo olvidadas, se convierten en puertas traseras para los atacantes.

# Análisis del Lado del Cliente y Acceso Indebido

## Fugas en la Lógica de Cliente

Analizar archivos JavaScript estáticos puede revelar parámetros ocultos, tokens de prueba o endpoints no documentados que pueden ser manipulados para obtener acceso no autorizado o información sensible.





## Unauthorized Method Access

Un riesgo crítico ocurre cuando un endpoint diseñado para, por ejemplo, solo aceptar peticiones GET, permite inesperadamente un DELETE debido a una validación insuficiente. Esto puede llevar a la eliminación de datos o a la alteración del sistema.

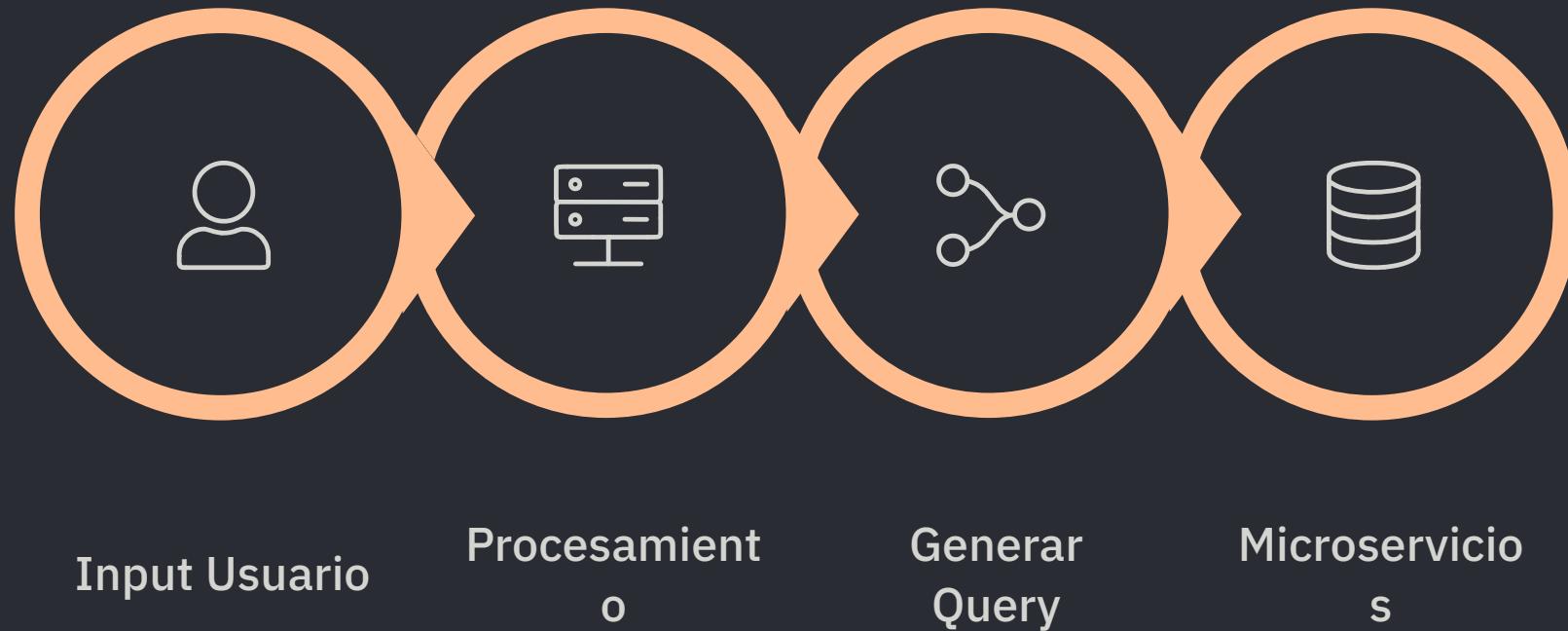
# Server-Side Parameter Pollution (SSPP)

## Definición Técnica de SSPP

La Contaminación de Parámetros en el Lado del Servidor (SSPP) ocurre cuando el backend procesa múltiples instancias de un mismo parámetro de maneras inesperadas, "ensuciando" sus propias consultas internas a otros servicios o bases de datos.



# Mecánica de la Query Interna



El servidor concatena el input del usuario para comunicarse con otros microservicios, bases de datos o sistemas internos. Una manipulación astuta de estos inputs puede llevar a la inyección de parámetros.

# Técnicas de Explotación de SSPP

1

## Inyección de Delimitadores

Uso de caracteres especiales como & y = para añadir parámetros adicionales a una consulta interna que el usuario no debería controlar directamente, alterando la lógica del servidor.

2

## Truncamiento con Fragmentos

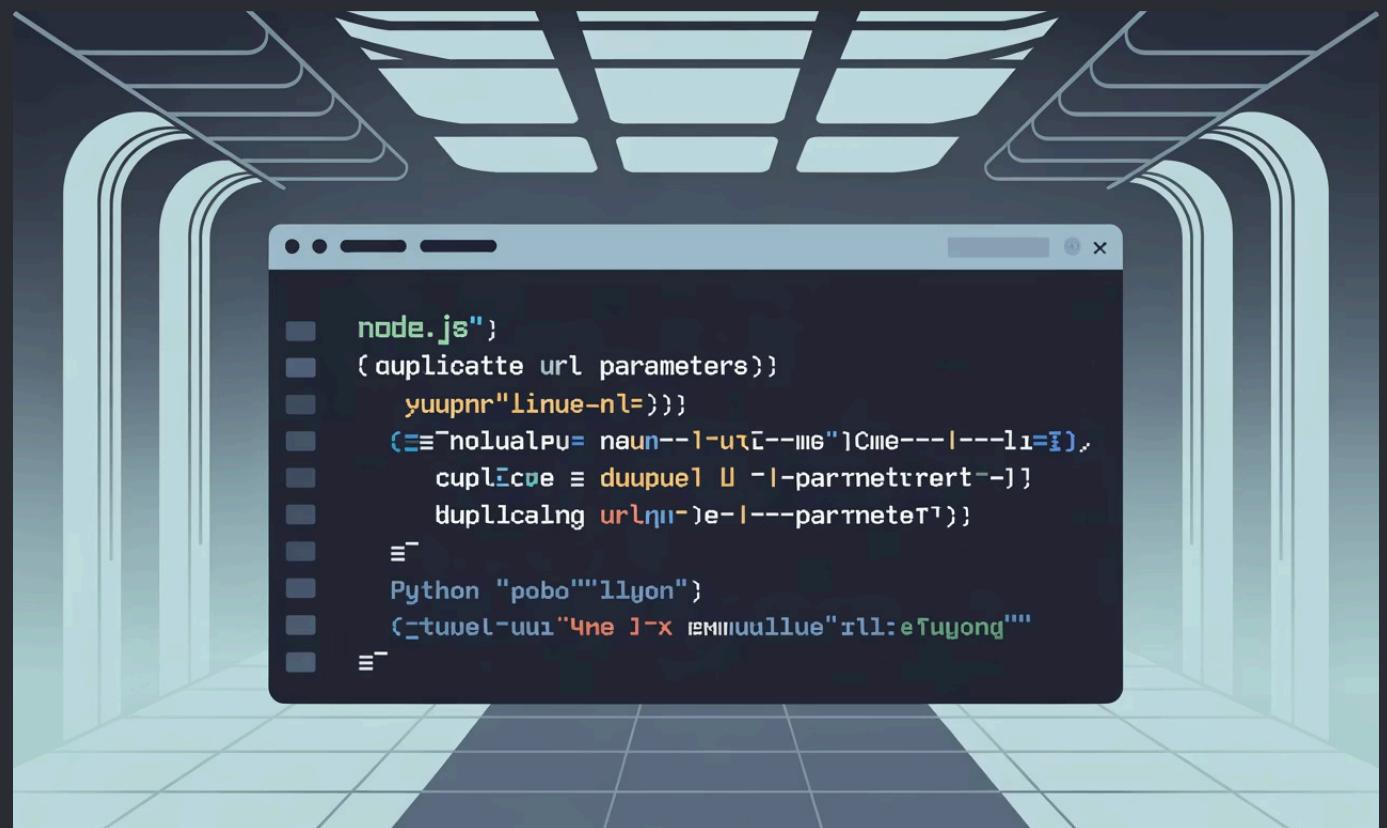
Empleo del fragmento URL (#) o su codificación (%23) para forzar al servidor a ignorar el resto de la consulta original, permitiendo al atacante redefinir la petición interna.

# Precedencia de Parámetros

## Comportamiento Inconsistente

Los diferentes lenguajes y frameworks de servidor manejan los parámetros duplicados de forma distinta. Por ejemplo, Node.js tiende a leer el último parámetro duplicado, mientras que Python o PHP pueden preferir el primero.

Comprender estas diferencias es clave para explotar SSPP, ya que permite al atacante sobrescribir o añadir valores a parámetros internos con éxito.



# Impacto del SSPP en la Seguridad



## Escalada de Privilegios

Manipulando parámetros internos, un atacante podría elevar sus permisos dentro de la aplicación, obteniendo acceso a funciones o datos restringidos.



## Bypass de Filtros de Seguridad

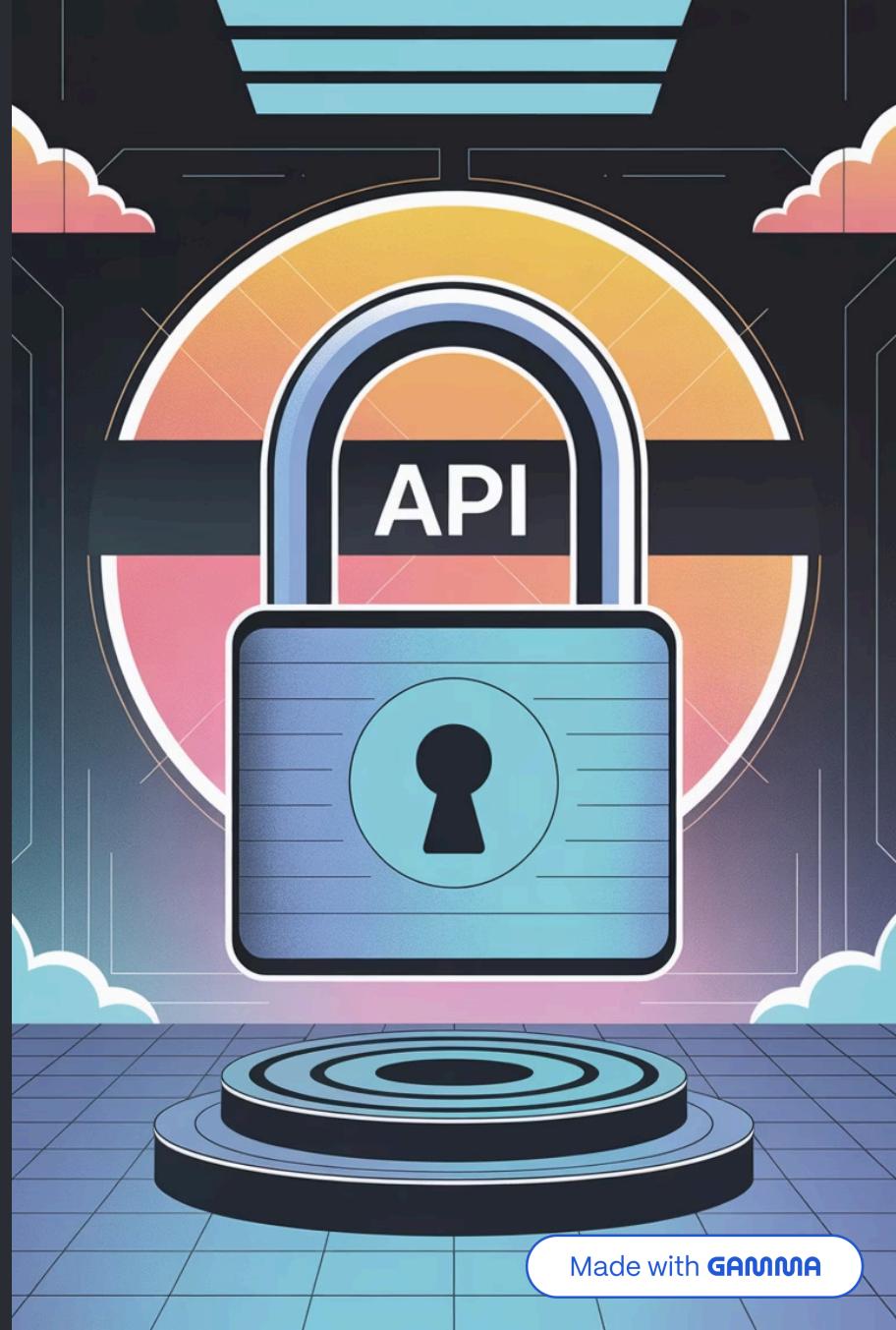
Al inyectar o modificar parámetros de forma encubierta, es posible evadir filtros de seguridad y controles de acceso que dependen de la validación de parámetros visibles.



## Compromiso de Datos

El SSPP puede llevar a la manipulación de consultas a bases de datos, resultando en la divulgación, modificación o eliminación no autorizada de información sensible.

# Ciberseguridad en APIs: Explorando la Integridad y Defensa de Objetos



# Mass Assignment: El Peligro del "Auto-binding"

El "Mass Assignment" es una vulnerabilidad donde un atacante puede modificar propiedades de un objeto que no deberían ser alteradas directamente por el cliente. Los frameworks modernos, con su énfasis en la productividad, a menudo facilitan el "auto-binding" de datos de entrada a modelos de objetos, lo que puede llevar a esta exposición no intencionada.



IMPACTO

# Inyección de Atributos JSON

Esta vulnerabilidad permite a los atacantes injectar campos inesperados en las peticiones JSON, potencialmente elevando privilegios o manipulando la lógica de la aplicación.

1

Añadir "role": "admin"

Un atacante podría añadir este campo para obtener permisos de administrador.

2

Modificar "price": 0

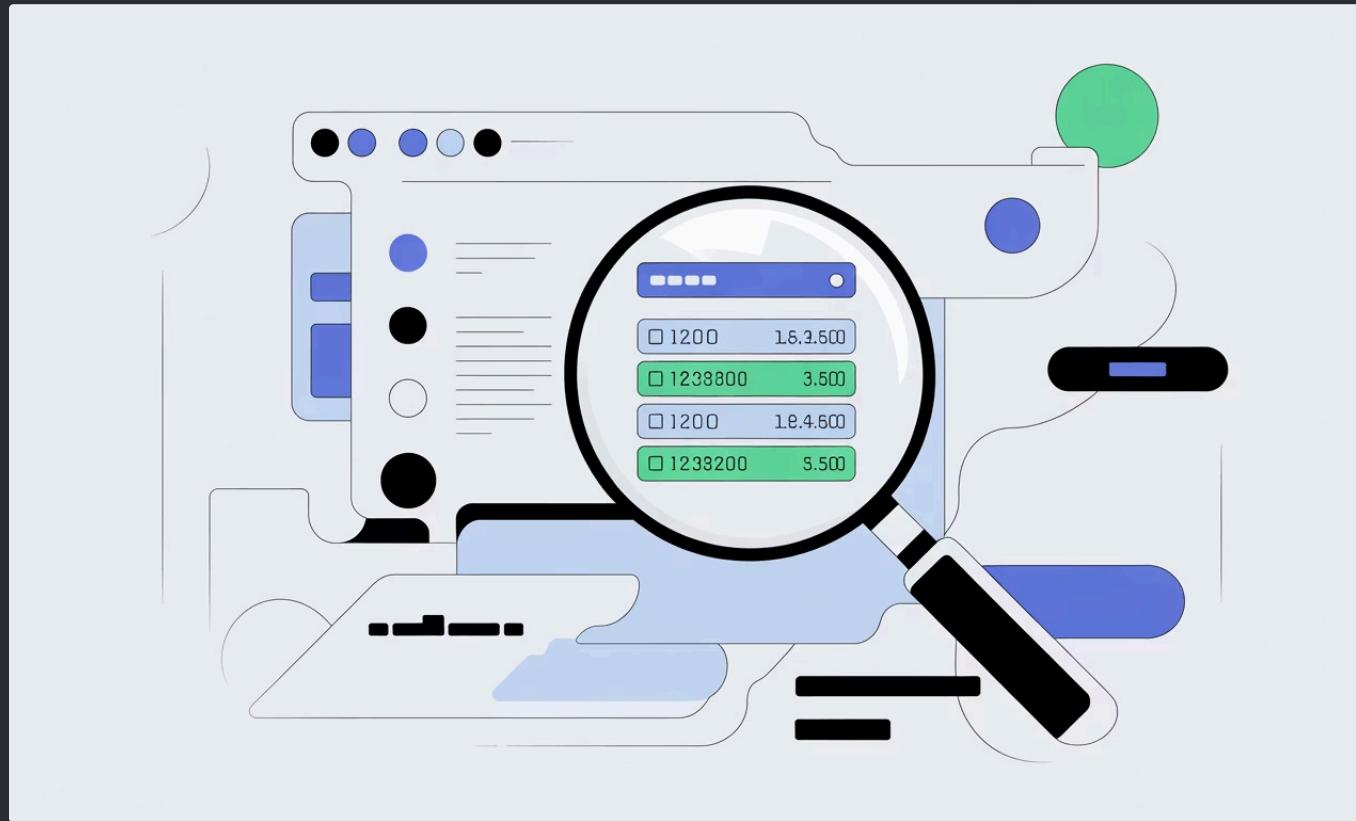
Cambiar valores sensibles en transacciones financieras.

3

Activar "feature": true

Habilitar funciones ocultas o en desarrollo.

# Identificación de Atributos Sensibles



Analizando cuidadosamente las respuestas del servidor, los atacantes pueden inferir qué campos de la base de datos son editables. Esta técnica se basa en la exposición accidental de información en las respuestas API, como el ID del usuario, estados de cuenta, o datos de acceso, que pueden ser posteriormente manipulados para comprometer el sistema.

# Manipulación de Lógica de Negocio

Una vulnerabilidad crítica donde la alteración de valores en el JSON puede modificar el flujo de dinero, aplicar descuentos indebidos o escalar privilegios, resultando en pérdidas económicas o acceso no autorizado.

Entrada Usuario

JSON Manipulado

Lógica Explotada

Impacto Negativo

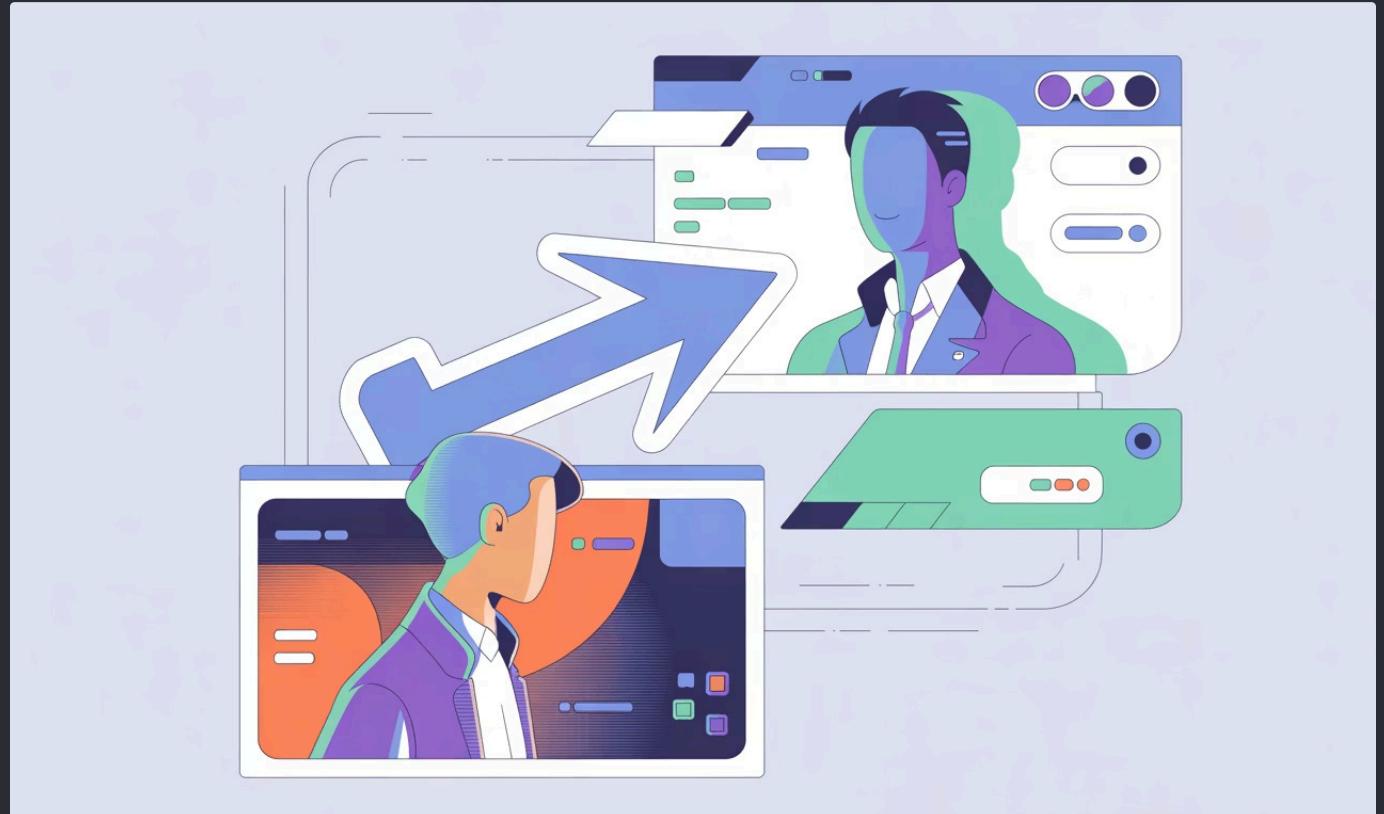
Este tipo de ataque explota la confianza implícita en los datos proporcionados por el cliente, afectando directamente la integridad operativa de la aplicación.

ACCESO NO AUTORIZADO

# Broken Object Level Authorization (BOLA)

## El Acceso Indebido

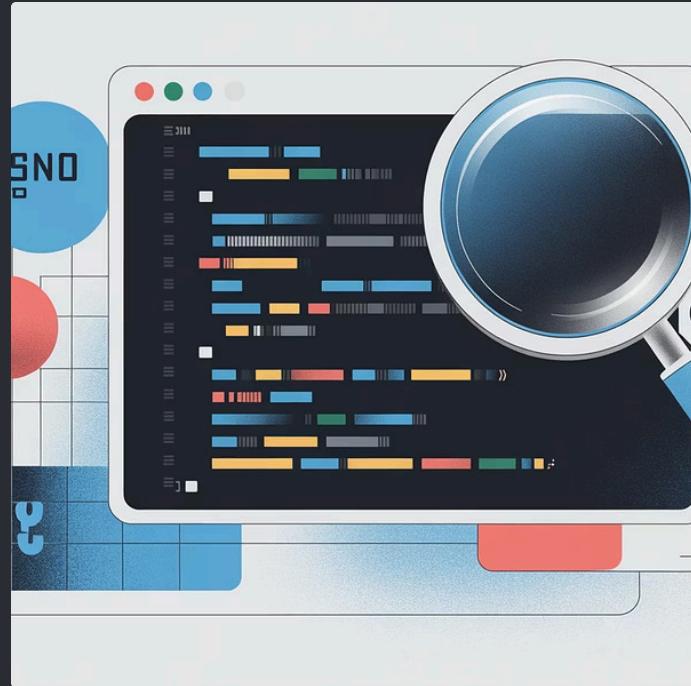
BOLA ocurre cuando un usuario puede acceder o modificar recursos que pertenecen a otro usuario simplemente cambiando el identificador del objeto en la petición API.



Esta falla es común en APIs que no implementan controles de autorización adecuados a nivel de objeto, permitiendo que un atacante eluda las restricciones de acceso.

# Fuzzing de Llaves JSON

El fuzzing de llaves JSON implica el uso de técnicas automatizadas para adivinar nombres de parámetros ocultos que el backend podría estar aceptando, exponiendo funcionalidades no documentadas o sensibles.



## Exploración

Intentar combinaciones comunes y nombres de atributos predecibles.



## Automatización

Usar herramientas para generar y probar un gran volumen de posibles llaves.



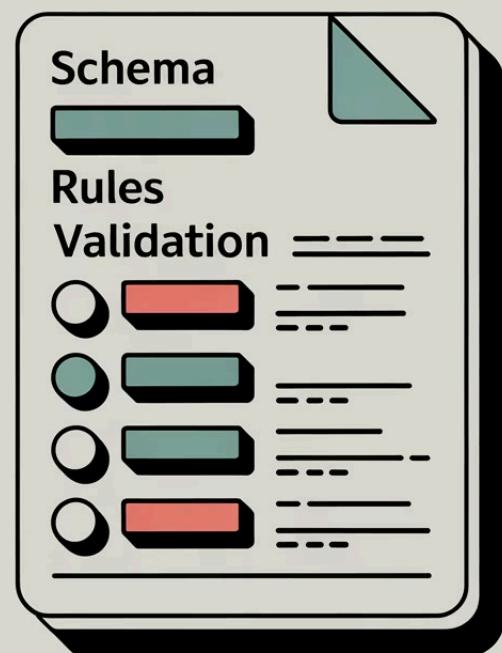
## Descubrimiento

Identificar respuestas inusuales que indiquen la aceptación de un atributo oculto.

# Mitigación Técnica: DTOs y Validación de Esquemas

## Data Transfer Objects (DTOs)

Los DTOs filtran los datos, asegurando que solo los campos permitidos se transfieran entre capas de la aplicación, previniendo el Mass Assignment.



## Validación de Esquemas

Definir y validar estrictamente los tipos de datos y los campos obligatorios en cada petición API es crucial para mantener la integridad de los datos.

# Seguridad en el Ciclo de Vida: DevSecOps para APIs

Integrar la auditoría de APIs desde las primeras etapas del desarrollo (DevSecOps) es fundamental para construir APIs seguras por diseño.

01

## Diseño Seguro

Considerar la seguridad desde la concepción de la API.

02

## Pruebas Continuas

Realizar pruebas de seguridad automatizadas y manuales.

03

## Monitoreo Activo

Vigilar la API en producción para detectar anomalías.

04

## Respuesta Rápida

Establecer planes para responder eficazmente a incidentes de seguridad.



# Conclusiones Clave y Pasos Siguientes

La seguridad de las APIs es un pilar fundamental en la protección de los sistemas modernos.

- **Priorizar la Validación:**
- **Autorización Fuerte:**
- **Adoptar DevSecOps:**
- **Mantenerse Informado:**