Pedro Garvich

Seguridad en funcionalidades de gestión de reservas en Restful Booker

Documento de Testing de Software

Versión 1.0.0

13/12/2025

Índice de Contenidos

1. Descripción general del Sistema

Restful-Booker es una API REST de ejemplo diseñada específicamente como un entorno de práctica (sandbox) para el aprendizaje y aplicación de pruebas de API (API Testing). Simula un sistema básico de reservas de hotel (*Bookings*), exponiendo los *endpoints* necesarios para realizar las operaciones fundamentales del protocolo CRUD (Crear, Leer, Actualizar y Eliminar reservas). Su arquitectura incluye un sistema de autenticación basado en *tokens* o *Basic Auth* requerido para las operaciones de modificación, y posee *endpoints* clave para la gestión de reservas y un *HealthCheck* (/ping).

2. Descripción de funcionalidades

Get Booking IDs

Devuelve los ids de todas las reservas existentes en la API. Opcionalmente puede recibir parámetros de filtrado como nombre de quién reserva y fechas de check in y check out.

Get Booking

Devuelve la información completa de una reserva basada en el id provisto.

Post Create Booking

Crea una nueva reserva en la API. Recibe un body en JSON o XML con los parámetros de la reserva. Ejemplo en JSON:

```
{    "firstname" : "Marco",
    "lastname" : "Polo",
    "totalprice" : 222,
    "depositpaid" : false,
    "bookingdates" : {
        "checkin" : "2018-01-01",
        "checkout" : "2019-01-01"
    },
    "additionalneeds" : "Breakfast" }
```

Delete Booking

Elimina una reserva previamente creada. Requiere un token de autenticación bálido o un Basic Auth Header.

3. Planificación y enfoque de Testing

3.1 Objetivo del Testing

El objetivo de este ciclo de pruebas es evaluar la seguridad de la API Restful-Booker, enfocándose en la detección de vulnerabilidades comunes en APIs REST, tales como inyecciones SQL, fallas en la validación de entradas, problemas de codificación de salida y controles de acceso insuficientes.
El testing se orienta principalmente a identificar riesgos asociados al procesamiento de datos controlados por el usuario y a la protección de recursos sensibles.

3.2 Alcance

Las pruebas se realizaron sobre los siguientes endpoints expuestos por la API:

- GET /booking
- GET /booking/{id}
- POST /booking
- DELETE /booking/{id}
- Endpoint de autenticación (para pruebas de control de acceso)

Dentro del alcance se incluyen:

- Pruebas de SQL Injection (boolean-based, error-based y time-based.
- Pruebas de validación y sanitización de input
- Pruebas de codificación segura de output
- Pruebas de control de acceso y autorización

Fuera del alcance:

- Pruebas de performance y stress
- Pruebas de lógica de negocio compleja
- Pruebas de infraestructura o configuración del servidor

3.3 Enfoque de Testing

Se utilizó un enfoque de testing de seguridad dinámico (DAST), realizando pruebas manuales mediante la herramienta Postman.
Los casos de prueba fueron diseñados siguiendo:

- Buenas prácticas de OWASP (OWASP Top 10 y OWASP API Security Top 10)
- Técnicas clásicas de detección de vulnerabilidades en APIs
- Comparación de respuestas entre requests legítimos y maliciosos

3.4 Criterios de Ejecución

Para cada caso de prueba se evaluaron los siguientes criterios:

- Código de estado HTTP
- Contenido del cuerpo de la respuesta
- Presencia de mensajes de error internos
- Comportamientos anómalos ante payloads maliciosos
- Consistencia entre respuestas esperadas y obtenidas

3.5 Criterios de Aprobación / Rechazo

Un caso de prueba se considera:

- **Pass** cuando la API responde de forma segura y consistente ante el escenario probado.
- **Fail / Defectuoso** cuando se detecta una vulnerabilidad o comportamiento inseguro que podría ser explotado en un entorno real.

4. Casos de Prueba

4.1 Vulnerabilidades SQLi

Este bloque verifica que la API no presente vulnerabilidades a inyecciones SQL.

3.1.1 Detección de SQL Injection Boolean-based en GET /booking

IDENTIFICADOR	SQLi-001
DESCRIPCIÓN	Verificar si el endpoint GET /booking es vulnerable a SQL Injection boolean-based, evaluando si variaciones lógicas en los parámetros producen cambios anómalos en las respuestas.
PRECONDICIONES	● Postman instalado. ● Entorno Postman con variable: base_url = https://restful-booker.herokuapp.com ● Scripts de test configurados en los requests A, B y C. ● Base de datos poblada previamente con bookings válidos (ver sección Setup). ● Endpoint /booking operativo.

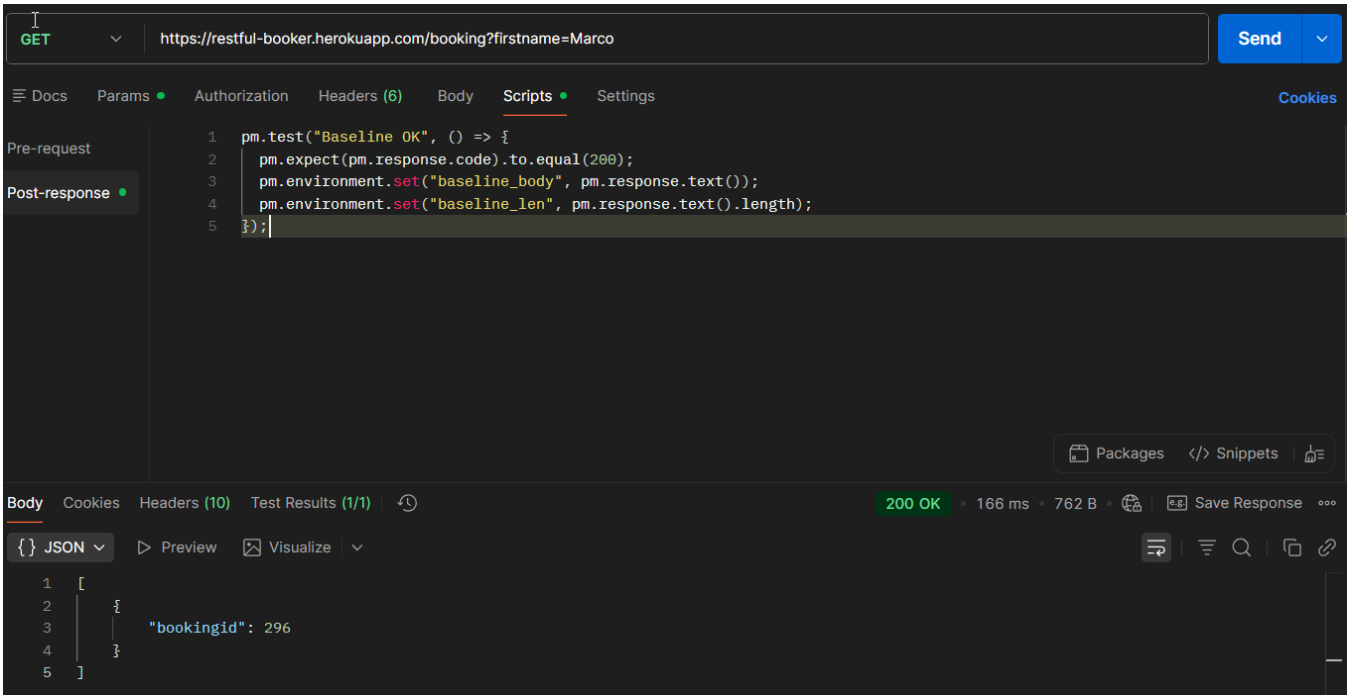
SET-UP DE DATOS	La API puede devolver [] si no hay reservas, lo cual impediría detectar variaciones entre TRUE y FALSE. Por eso se deben crear al menos 3 reservas antes de ejecutar la prueba.
	Request POST de ejemplo
	POST {{base_url}}/booking
	Body (JSON):
	{ "firstname": "Marco",
	"lastname": "Brown",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2018-01-01",
	"checkout": "2019-01-01" },
	"additionalneeds": "Breakfast" }
	Test asociado al POST para verificar la creación de los datos:
	pm.test("Status code is 200", function () {
	pm.response.to.have.status(200); });
	pm.test("Verifying booking id", function () {
	const jsonData = pm.response.json();
	const bookingId = jsonData.bookingid;
	pm.environment.set("BookingId", bookingId);
	pm.expect(bookingId).to.not.be.null; });

DATOS DE ENTRADA	A (baseline) firstname: Marco (o cualquier existente) B (TRUE) firstname: ‘ OR ‘1’=’1 C (FALSE firstname ‘ AND ‘1’=’2
PASOS A SEGUIR	● Ejecutar Request A — Baseline Enviar GET {{base_url}}/booking?firstname=Marco. Confirmar que el status es 200. El script guarda baseline_body y baseline_len ● Ejecutar Request B — Inyección TRUE Enviar GET {{base_url}}/booking?firstname=' OR '1'='1. Verificar ausencia de errores SQL. El script guarda inj_true_len ● Ejecutar Request C — Inyección FALSE Enviar GET {{base_url}}/booking?firstname=' AND '1'='2. Verificar ausencia de errores SQL. El script guarda inj_false_len

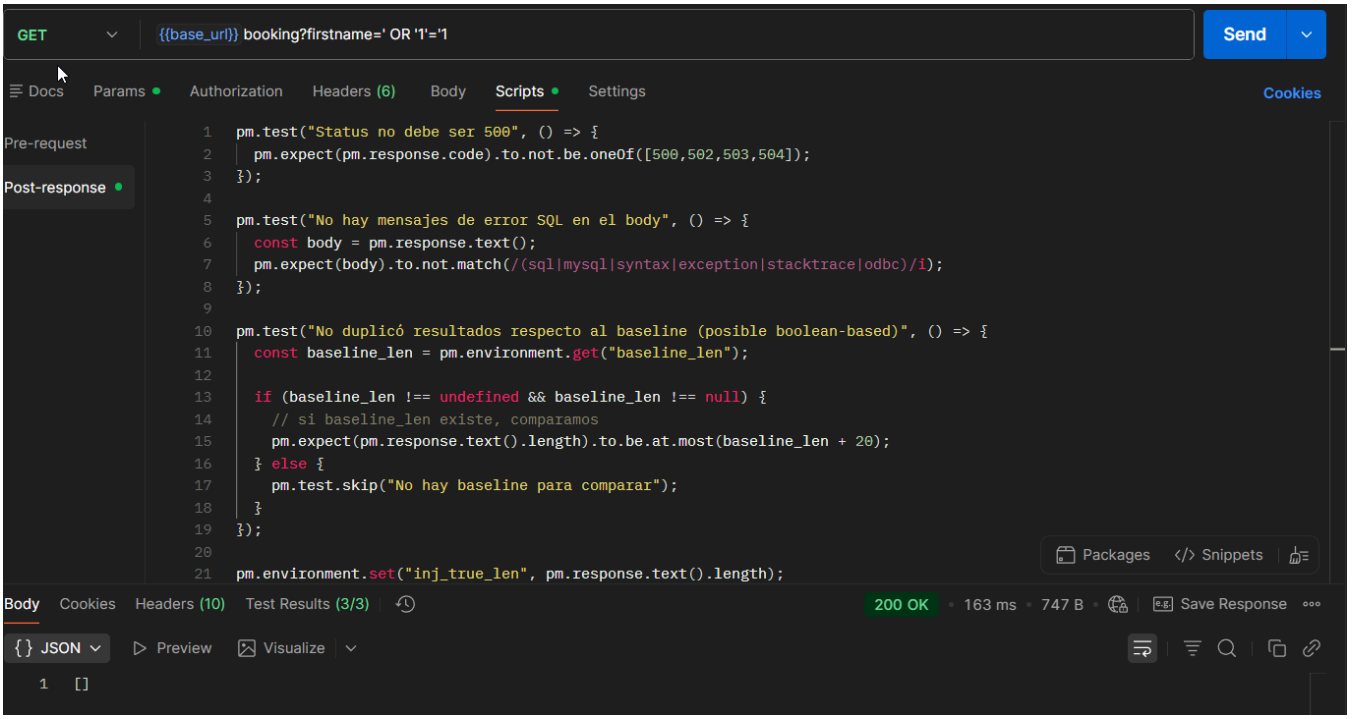
	Postman ejecuta el test que compara TRUE vs FALSE (tolerancia: ±20).
RESULTADO ESPERADO	Status code debe ser 200 o 400, nunca 500/502/503/504. El body no debe contener indicadores de error SQL: sql, mysql, syntax, exception, stacktrace, odbc, etc. La respuesta del request B (TRUE) no debe diferir significativamente de C (FALSE). La longitud de ambas respuestas debe ser similar (±200). No debe observarse aumento de registros en B ni reducción en C. B y C similares o diferencias mínimas
RESULTADO OBTENIDO	A: [{"bookingid": 296}] B: [] C: []
ESTADO	**Pass**

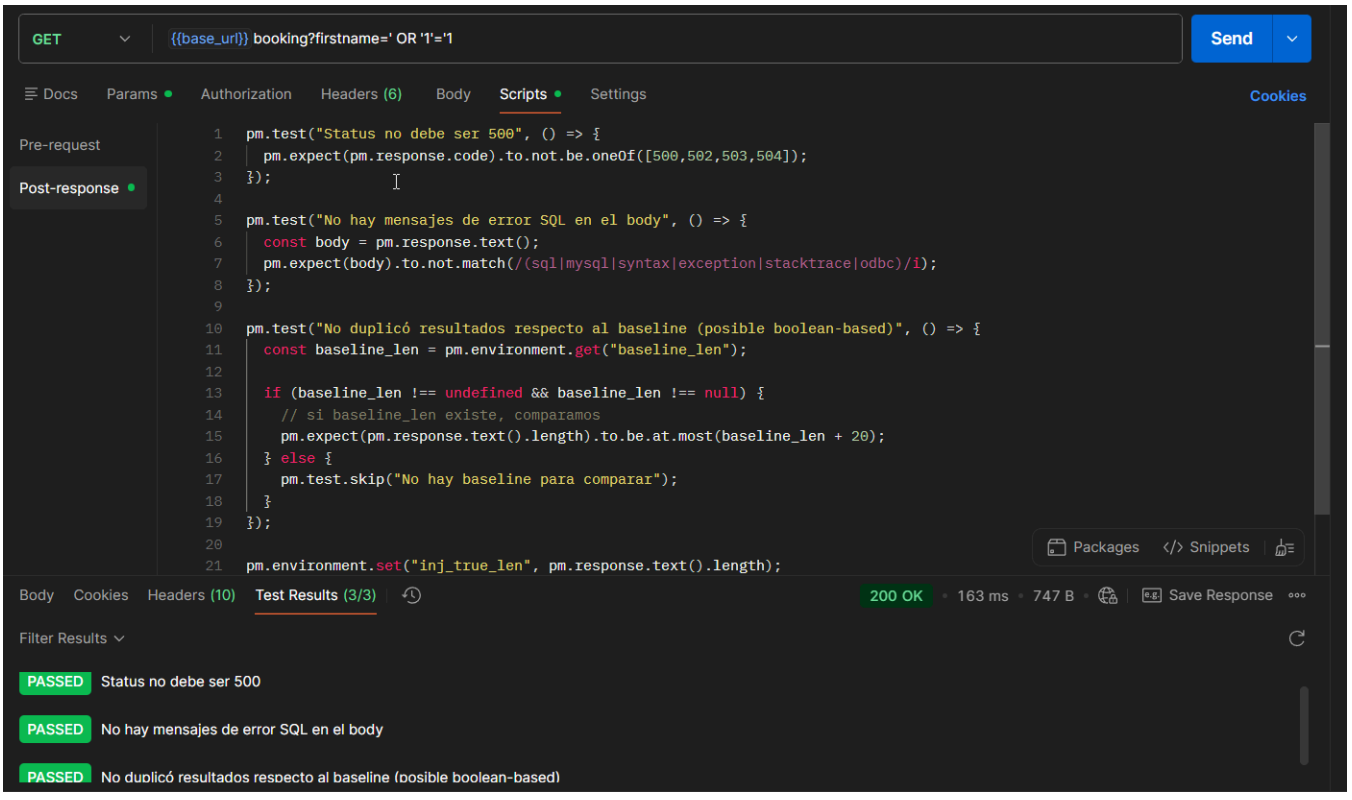
Evidencias de la prueba

GET A (BaseLine):

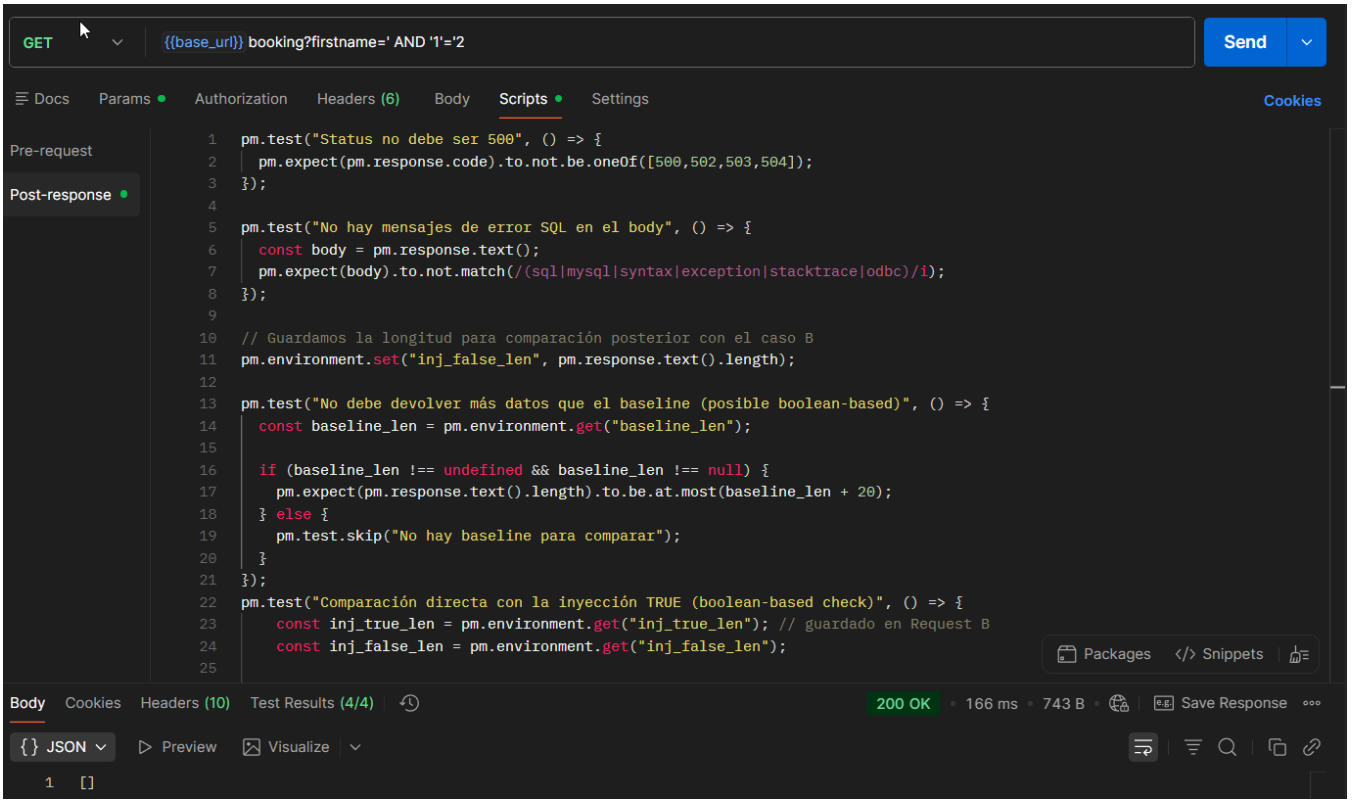


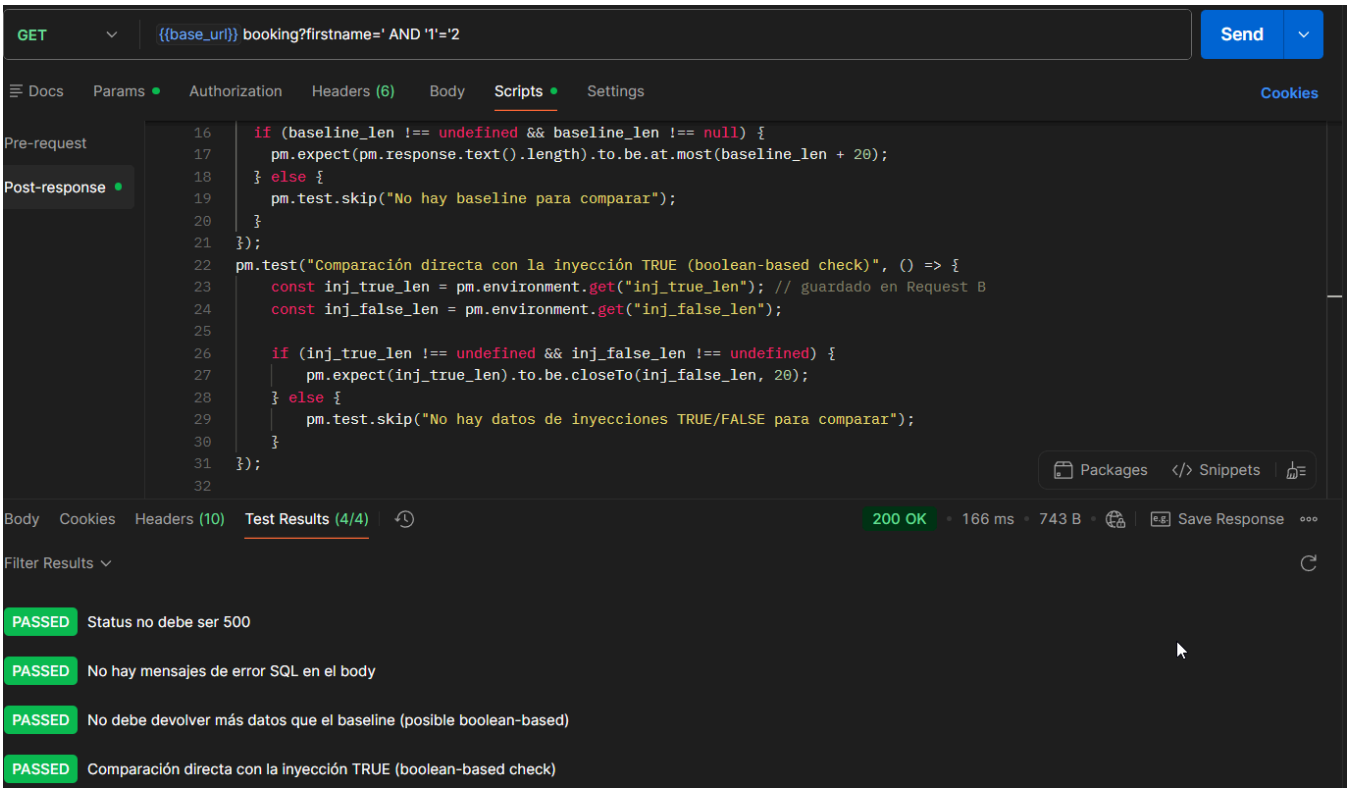
GET B:


GET
{{base_url}} booking?firstname=' OR '1'='1
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status no debe ser 500", () => {
pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
});
pm.test("No hay mensajes de error SQL en el body", () => {
const body = pm.response.text();
pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
});
pm.test("No duplicó resultados respecto al baseline (posible boolean-based)", () => {
const baseline_len = pm.environment.get("baseline_len");
if (baseline_len !== undefined && baseline_len !== null) {
// si baseline_len existe, comparamos
pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
} else {
pm.test.skip("No hay baseline para comparar");
}
});
pm.environment.set("inj_true_len", pm.response.text().length);
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (3/3)
200 OK
163 ms
747 B
Save Response
{} JSON
Preview
Visualize
1 []

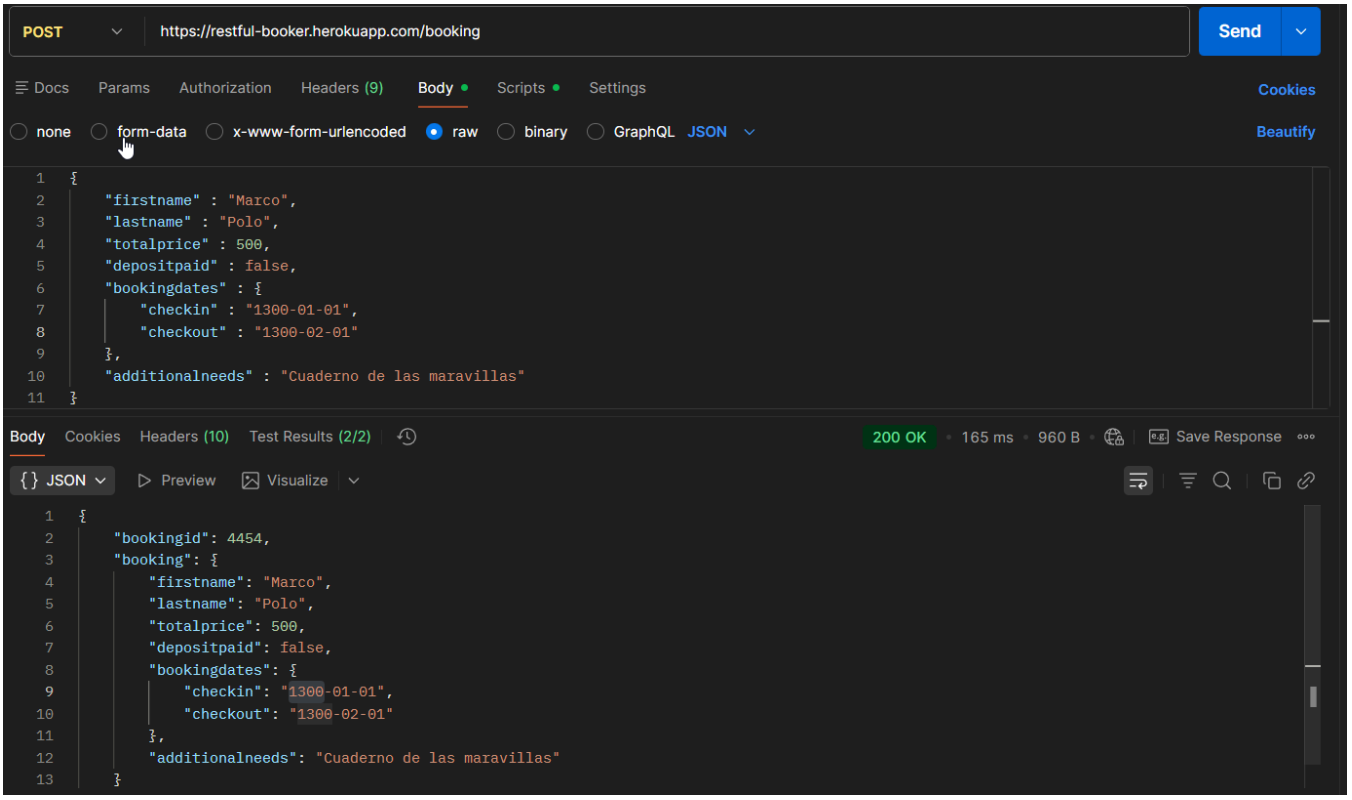

GET
{{base_url}} booking?firstname=' OR '1'='1
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status no debe ser 500", () => {
pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
});
pm.test("No hay mensajes de error SQL en el body", () => {
const body = pm.response.text();
pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
});
pm.test("No duplicó resultados respecto al baseline (posible boolean-based)", () => {
const baseline_len = pm.environment.get("baseline_len");
if (baseline_len !== undefined && baseline_len !== null) {
// si baseline_len existe, comparamos
pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
} else {
pm.test.skip("No hay baseline para comparar");
}
});
pm.environment.set("inj_true_len", pm.response.text().length);
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (3/3)
200 OK
163 ms
747 B
Save Response
Filter Results
PASSED Status no debe ser 500
PASSED No hay mensajes de error SQL en el body
PASSED No duplicó resultados respecto al baseline (posible boolean-based)

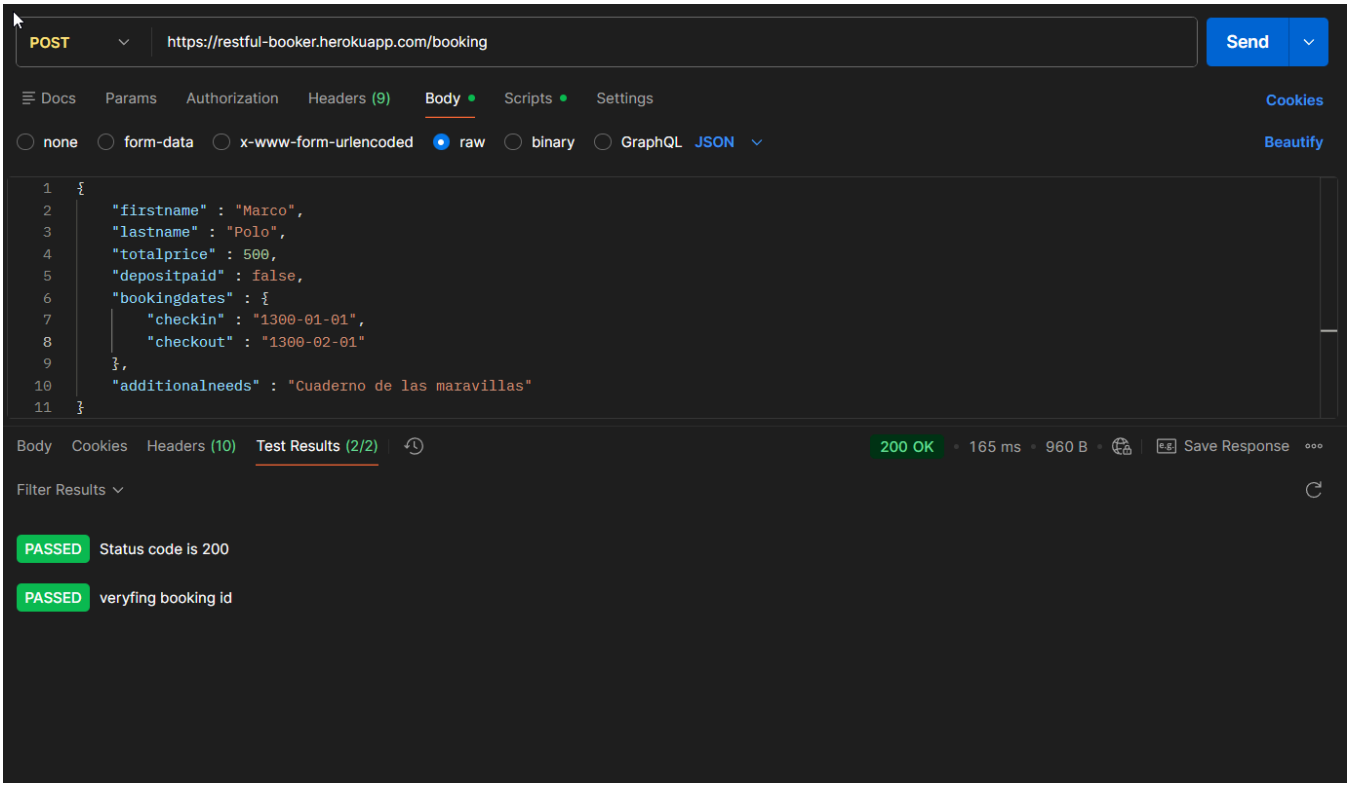
GET C:


GET
{{base_url}} booking?firstname=' AND '1'='2
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status no debe ser 500", () => {
pm.expect(pm.response.code).to.not.be.oneOf([500,502,503,504]);
});
pm.test("No hay mensajes de error SQL en el body", () => {
const body = pm.response.text();
pm.expect(body).to.not.match(/(sql|mysql|syntax|exception|stacktrace|odbc)/i);
});
// Guardamos la longitud para comparación posterior con el caso B
pm.environment.set("inj_false_len", pm.response.text().length);
pm.test("No debe devolver más datos que el baseline (posible boolean-based)", () => {
const baseline_len = pm.environment.get("baseline_len");
if (baseline_len !== undefined && baseline_len !== null) {
pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
} else {
pm.test.skip("No hay baseline para comparar");
}
});
pm.test("Comparación directa con la inyección TRUE (boolean-based check)", () => {
const inj_true_len = pm.environment.get("inj_true_len"); // guardado en Request B
const inj_false_len = pm.environment.get("inj_false_len");
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (4/4)
200 OK
166 ms
743 B
Save Response
{} JSON
Preview
Visualize
1 []


GET
{{base_url}} booking?firstname=' AND '1'='2
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
if (baseline_len !== undefined && baseline_len !== null) {
pm.expect(pm.response.text().length).to.be.at.most(baseline_len + 20);
} else {
pm.test.skip("No hay baseline para comparar");
}
});
pm.test("Comparación directa con la inyección TRUE (boolean-based check)", () => {
const inj_true_len = pm.environment.get("inj_true_len"); // guardado en Request B
const inj_false_len = pm.environment.get("inj_false_len");
if (inj_true_len !== undefined && inj_false_len !== undefined) {
pm.expect(inj_true_len).to.be.closeTo(inj_false_len, 20);
} else {
pm.test.skip("No hay datos de inyecciones TRUE/FALSE para comparar");
}
});
Packages
Snippets
Body
Cookies
Headers (10)
Test Results (4/4)
200 OK
166 ms
743 B
Save Response
Filter Results
PASSED
Status no debe ser 500
PASSED
No hay mensajes de error SQL en el body
PASSED
No debe devolver más datos que el baseline (posible boolean-based)
PASSED
Comparación directa con la inyección TRUE (boolean-based check)

SETUP:

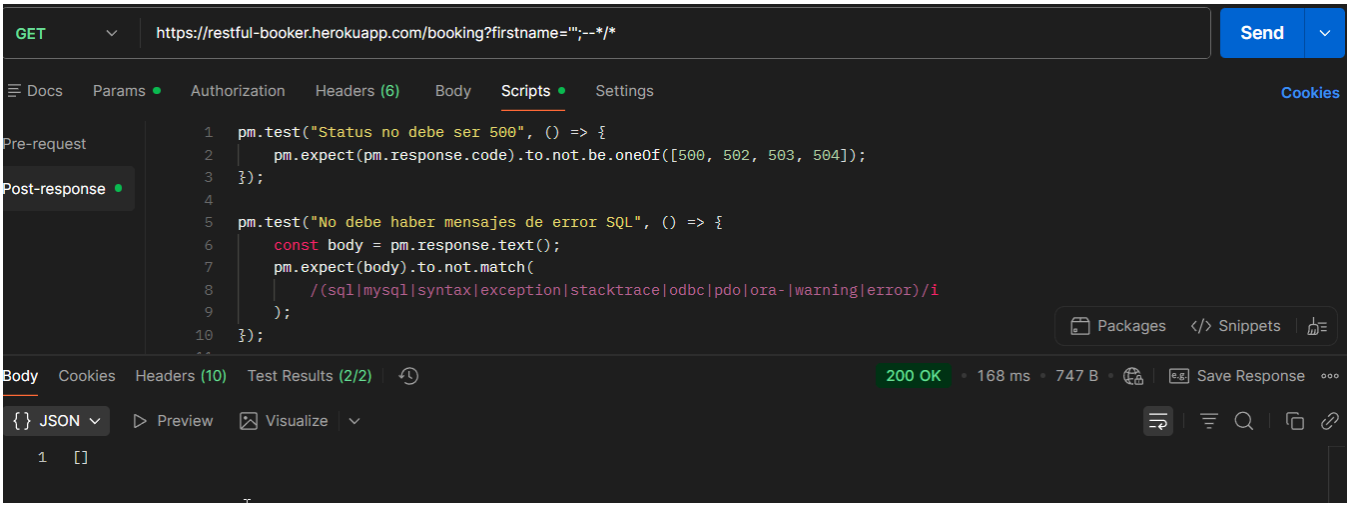

POST
https://restful-booker.herokuapp.com/booking
Send
Docs
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"firstname" : "Marco",
"lastname" : "Polo",
"totalprice" : 500,
"depositpaid" : false,
"bookingdates" : {
"checkin" : "1300-01-01",
"checkout" : "1300-02-01"
},
"additionalneeds" : "Cuaderno de las maravillas"
}
Body
Cookies
Headers (10)
Test Results (2/2)
200 OK
165 ms
960 B
Save Response
JSON
Preview
Visualize
{
"bookingid": 4454,
"booking": {
"firstname": "Marco",
"lastname": "Polo",
"totalprice": 500,
"depositpaid": false,
"bookingdates": {
"checkin": "1300-01-01",
"checkout": "1300-02-01"
},
"additionalneeds": "Cuaderno de las maravillas"
}


POST
https://restful-booker.herokuapp.com/booking
Send
Docs
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"firstname" : "Marco",
"lastname" : "Polo",
"totalprice" : 500,
"depositpaid" : false,
"bookingdates" : {
"checkin" : "1300-01-01",
"checkout" : "1300-02-01"
},
"additionalneeds" : "Cuaderno de las maravillas"
}
Body
Cookies
Headers (10)
Test Results (2/2)
200 OK
165 ms
960 B
Save Response
Filter Results
PASSED
Status code is 200
PASSED
veryfing booking id

3.1.2 Error-Based SQLi en GET /booking

IDENTIFICADOR	SQLi-002
DESCRIPCIÓN	Verificar si el endpoint GET /booking es vulnerable a SQL Injection Error-Based, enviando un parámetro con comillas malformadas para observar si la API revela errores internos del motor SQL (mensajes de error, stacktrace o status 500).
PRECONDICIONES	• Postman instalado. • El endpoint /booking operativo.
DATOS DE ENTRADA	firstname="'
PASOS A SEGUIR	• Abrir Postman y seleccionar método GET. • Enviar la solicitud a https://restful-booker.herokuapp.com/booking?firstname="';--*/* • Observar status code y body de respuesta
RESULTADO ESPERADO	• No debe devolverse un status 500/502/503/504. • No deben aparecer mensajes típicos de error SQL como sql, mysql, syntax, exception, stacktrace, odbc, pdo, ORA-. • La API debe responder con un 200 o 400 y un body seguro (generalmente []).
RESULTADO OBTENIDO	Status Code 200 Response Body: []
ESTADO	**Pass**

Evidencias de la prueba



3.1.3 SQLi Error Based Body — /booking

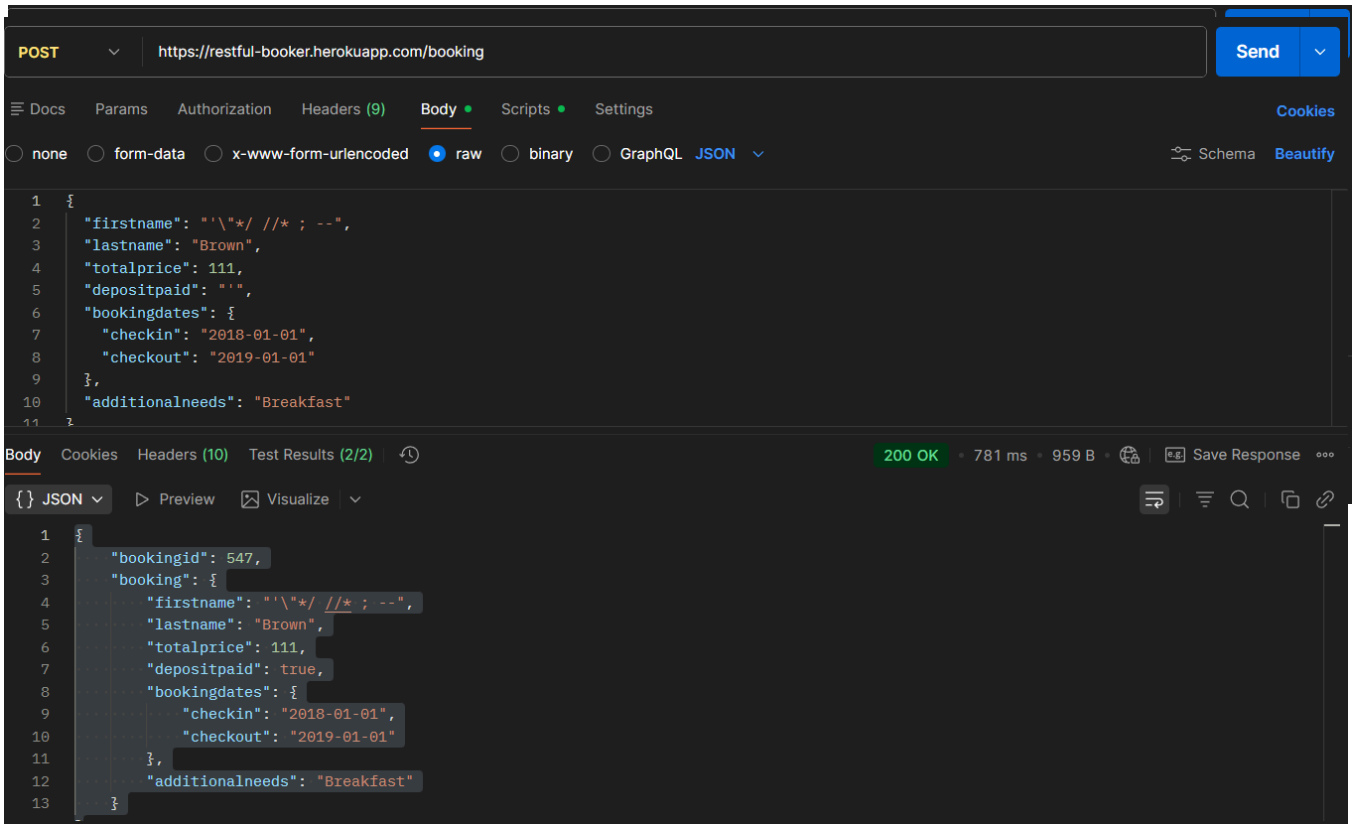
IDENTIFICADOR	SQLi-003
DESCRIPCIÓN	Evaluar si el endpoint **POST /booking** es vulnerable a SQL Injection de tipo **error-based**, enviando un payload diseñado para romper la sintaxis SQL mediante comillas simples, dobles y otros caracteres especiales. El objetivo es observar si el servidor expone mensajes de error internos o comportamientos inesperados que indiquen falta de sanitización de entrada.
PRECONDICIONES	API en funcionamiento.
DATOS DE ENTRADA	Body: { "firstname": "'\"*/ //* ; --", "lastname": "Brown", "totalprice": 111, "depositpaid": true, "bookingdates": { "checkin": "2018-01-01", "checkout": "2019-01-01" }, "additionalneeds": "Breakfast"}
PASOS A SEGUIR	● Abrir Postman y seleccionar POST /booking. ● URL: https://restful-booker.herokuapp.com/booking ● Ir a Body → raw → JSON. ● Pegar el payload malicioso y enviar la solicitud. ● Registrar el código de estado HTTP y el cuerpo de la respuesta.
RESULTADO ESPERADO	● El servidor no debe devolver errores SQL. ● No deben aparecer mensajes como SQL syntax error, mysql, exception, stacktrace, ORA-00933. ● El status code debe ser 200 o algún código de validación lógica (400), pero nunca 500+. ● El valor malicioso debe ser tratado como un texto más (si no hay sanitización del lado del servidor). ● El backend no debe mostrar evidencia de ejecutar SQL.

RESULTADO OBTENIDO	Status code 200
	Response body:
	{
	"bookingid": 547,
	"booking": {
	"firstname": "'\"*/ //* ; --",

	"lastname": "Brown",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2018-01-01",
	"checkout": "2019-01-01"
	},
	"additionalneeds": "Breakfast"
	}
	}

OBSERVACIONES	• La API acepta y almacena el payload malicioso sin filtrarlo. • No se generan errores SQL ni mensajes internos. • No se expone información sensible. • El comportamiento es consistente con un backend que no ejecuta SQL (no sabemos si Restful Booker lo hace).
ESTADO	**Pass**

Evidencias de la prueba:



3.1.4 SQLi Boolean-Based en Body

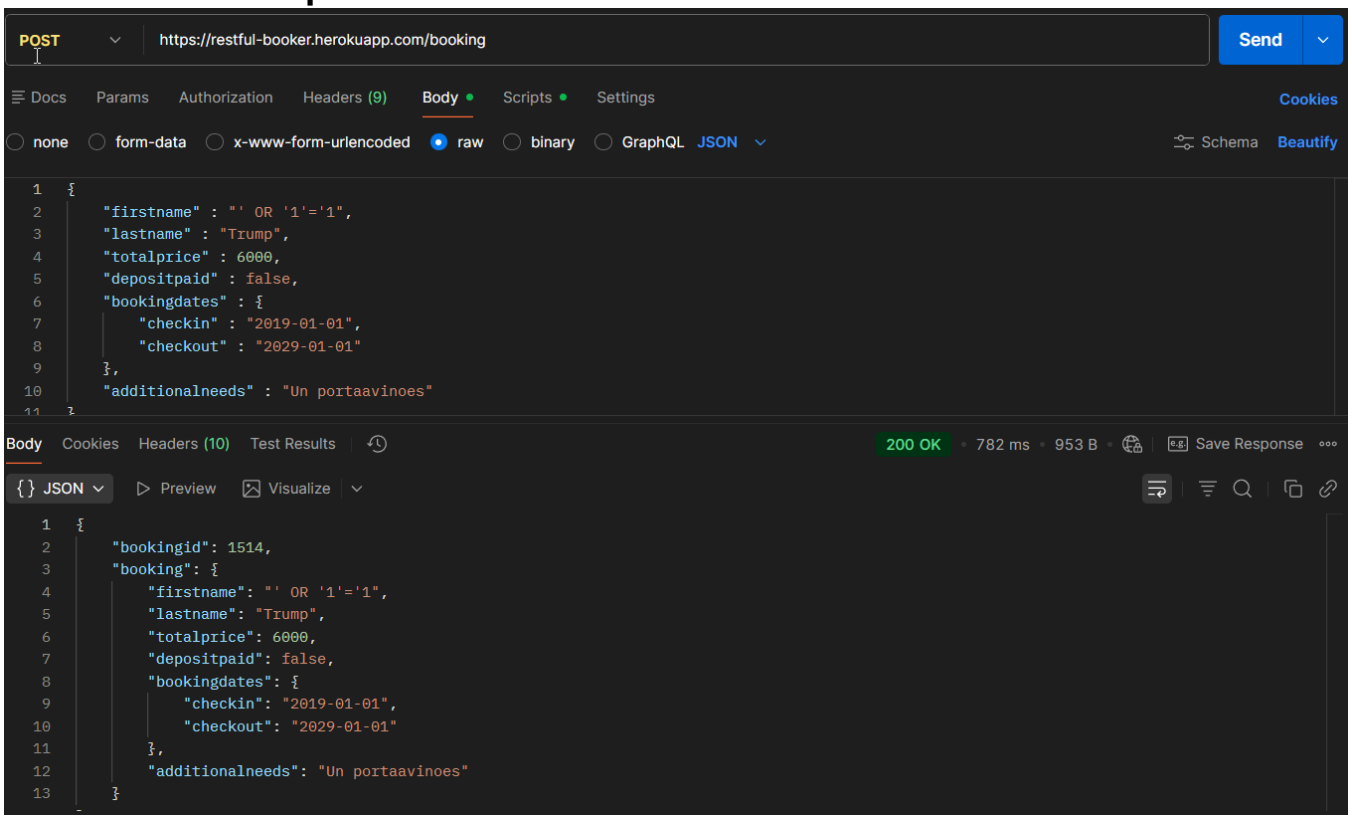
IDENTIFICADOR	SQLi-004
DESCRIPCIÓN	Evaluar si el endpoint **POST /booking** es vulnerable a SQL injection de tipo **boolean-based** comparando las respuestas a payloads que, interpretados como código SQL, serían siempre verdaderos o siempre falsos. Tipos o contenidos de respuestas significativamente distintas indicarían un patrón típico de SQL Injection inferencial y puede detectarse aunque no se expongan errores SQL.
PRECONDICIONES	API en funcionamiento.

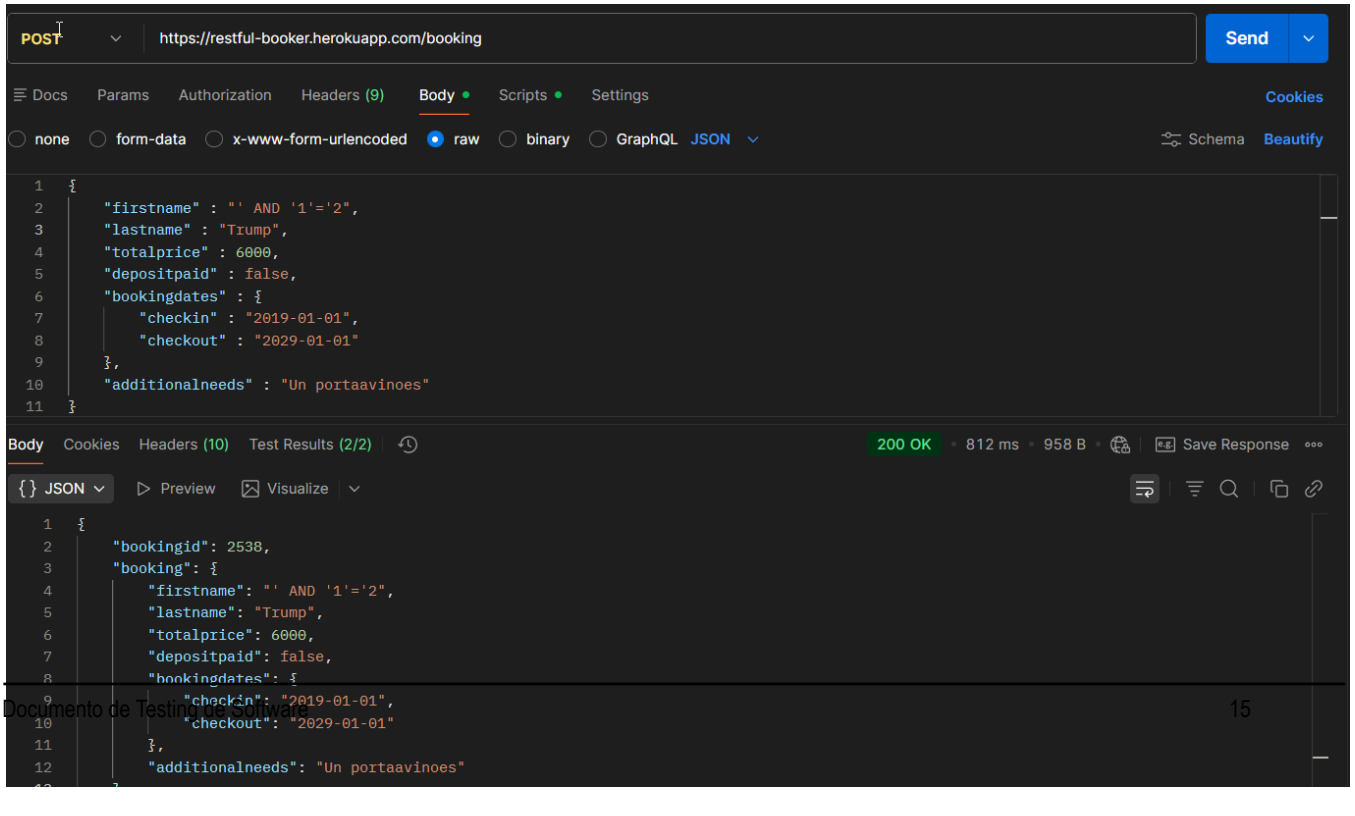
DATOS DE ENTRADA	{ "firstname" : "' OR '1'='1",
	"lastname" : "Trump",
	"totalprice" : 6000,
	"depositpaid" : false,
	"bookingdates" : {
	"checkin" : "2019-01-01",
	"checkout" : "2029-01-01"
	},
	"additionalneeds" : "Un portaavinoes"}
	Body en caso False:
	{ "firstname" : "' AND '1'='2",
	"lastname" : "Trump",
	"totalprice" : 6000,
	"depositpaid" : false,
	"bookingdates" : {
	"checkin" : "2019-01-01",
	"checkout" : "2029-01-01"
	},
	"additionalneeds" : "Un portaavinoes"
	}

PASOS A SEGUIR	● Enviar POST con el payload TRUE (‘ OR ‘1’=‘1) y guardar status code y body response ● Enviar POST con el payload FALSE (‘ OR ‘1’=‘2) y guardar status code y body response ● Comparar status code y body response de ambas request
RESULTADO ESPERADO	● Ambos requests deben devolver el mismo status code

	• El contenido de respuesta debe tener estructura equivalente • No deben aparecer errores SQL ni stack traces • Longitud de respuesta similar
RESULTADO OBTENIDO	• Status code 200 en ambos casos • Las longitudes idénticas • No aparecen errores SQL
ESTADO	**Pass**

Evidencias de la prueba:





3.1.5 SQL Injection Time-Based en POST Body

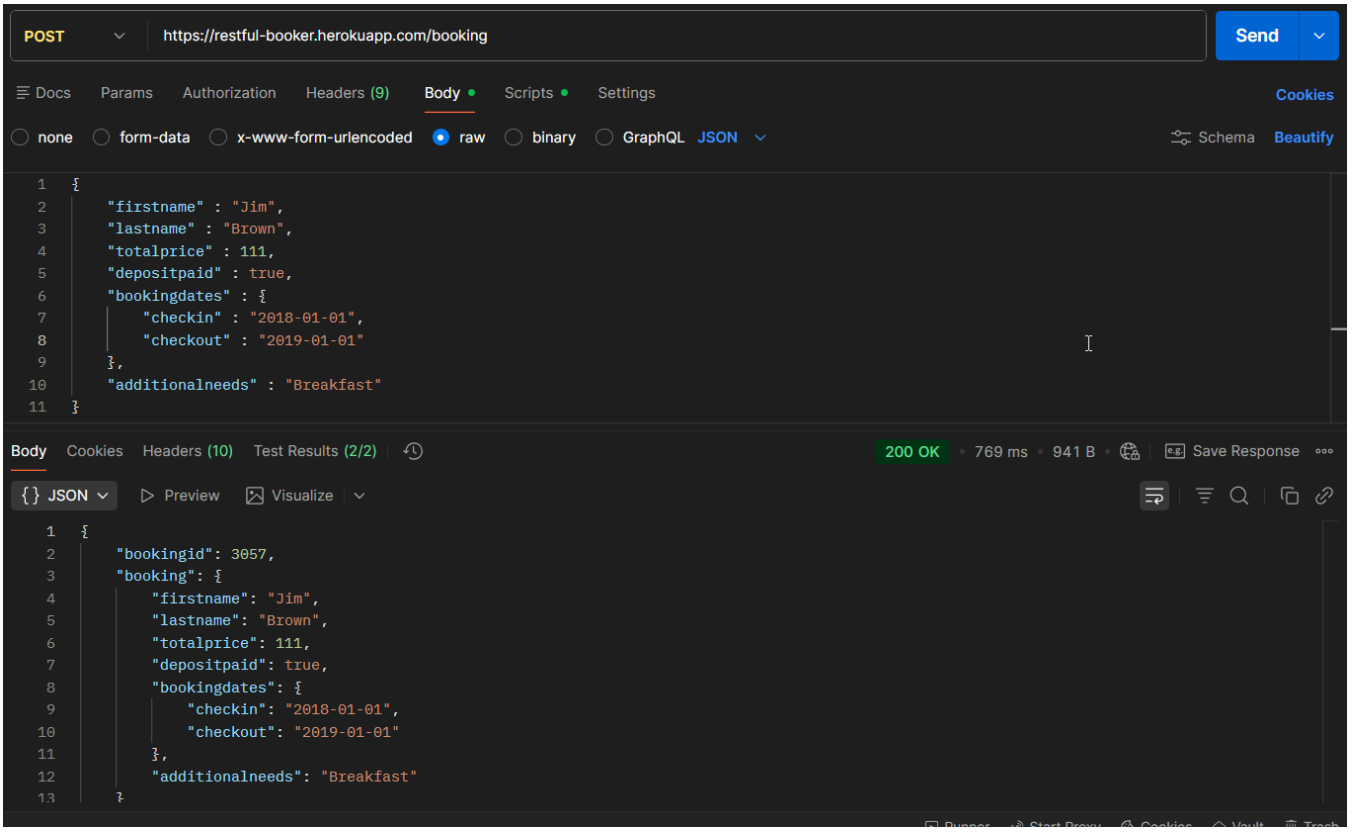
IDENTIFICADOR	SQLi-005
DESCRIPCIÓN	Esta prueba mide la diferencia de tiempo entre un request legítimo y uno que contiene un payload diseñado para introducir un retraso deliberado en el motor de base de datos. Con ello se podrá identificar una posible vulnerabilidad de SQL Injection Time-Based en el endpoint POST /booking.
PRECONDICIONES	API en funcionamiento. El tiempo de respuesta estándar del endpoint es estable (menos de 1000 ms).

DATOS DE ENTRADA	Body de Request Legítimo:
	{ "firstname": "Jim",
	"lastname": "Brown",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2024-01-01",
	"checkout": "2024-02-01"
	},
	"additionalneeds": "Breakfast"
	}
	Body malicioso:
	{ "firstname": "test'; SELECT pg_sleep(5)--",
	"lastname": "Brown",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2024-01-01",
	"checkout": "2024-02-01"
	},
	"additionalneeds": "Breakfast"
	}
	Payloads alternativos por motor SQL:
	MySQL test' OR SLEEP(5)-- -

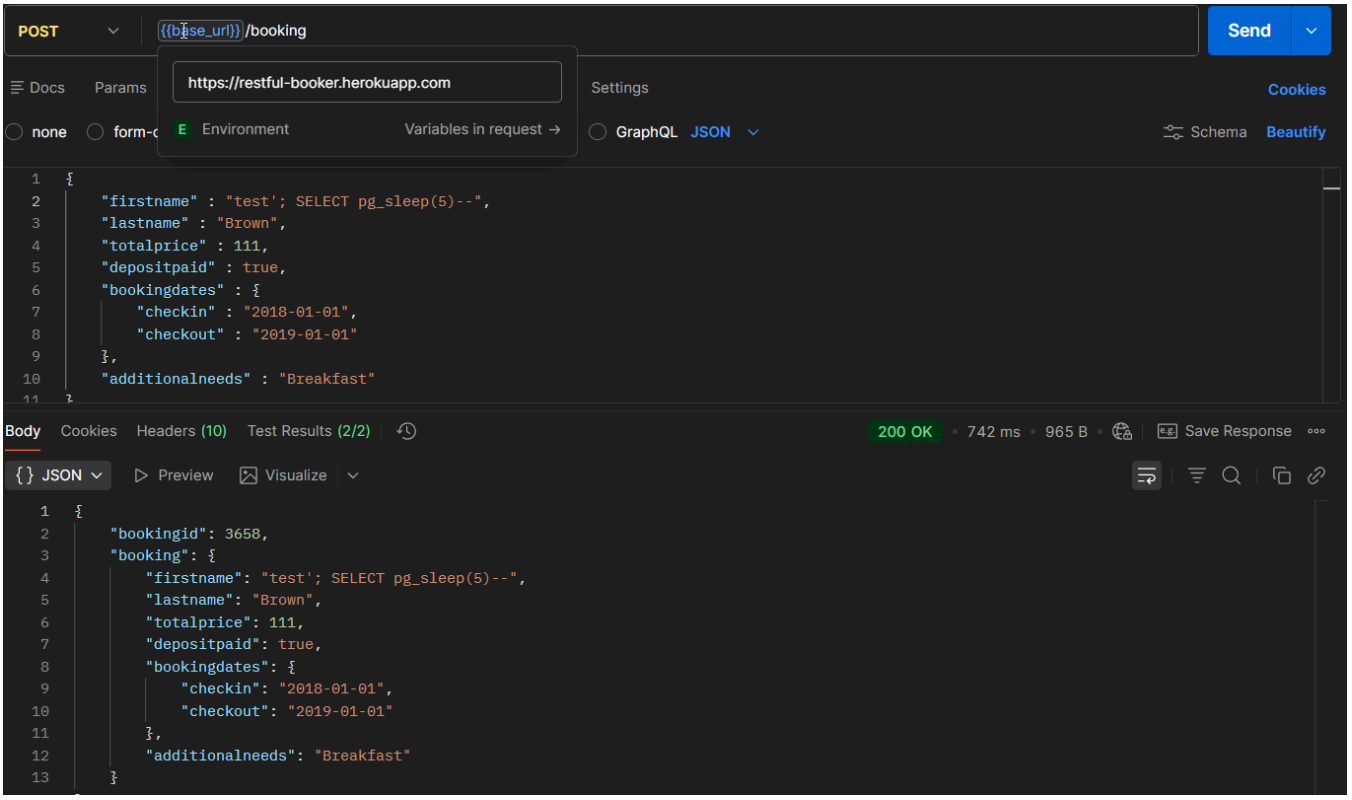
	PostgreSQL test'; SELECT pg_sleep(5)-- SQL Server test'; WAITFOR DELAY '0:0:5'--
PASOS A SEGUIR	• Enviar el body legítimo y registrar status code, body response y tiempo de respuesta. • Enviar payload de time-based SQLi reemplazando firstname por el payload malicioso (en las tres versiones) y registrar status code, body response y tiempo de respuesta. • Repetir el request malicioso 3 veces para confirmar consistencia, registrar todos los tiempos. • Comparar tiempo de respuesta de los request maliciosos con el del legítimo.
RESULTADO ESPERADO	• Status code en todos los casos 200 • Body similar al request normal • Tiempos de respuesta similar al request normal (diferencia <1000 ms)
RESULTADO OBTENIDO	• Status code en todos los casos 200 • Body similar al request normal • Tiempos de respuesta similar al request normal (diferencia <1000 ms)
ESTADO	**Pass**

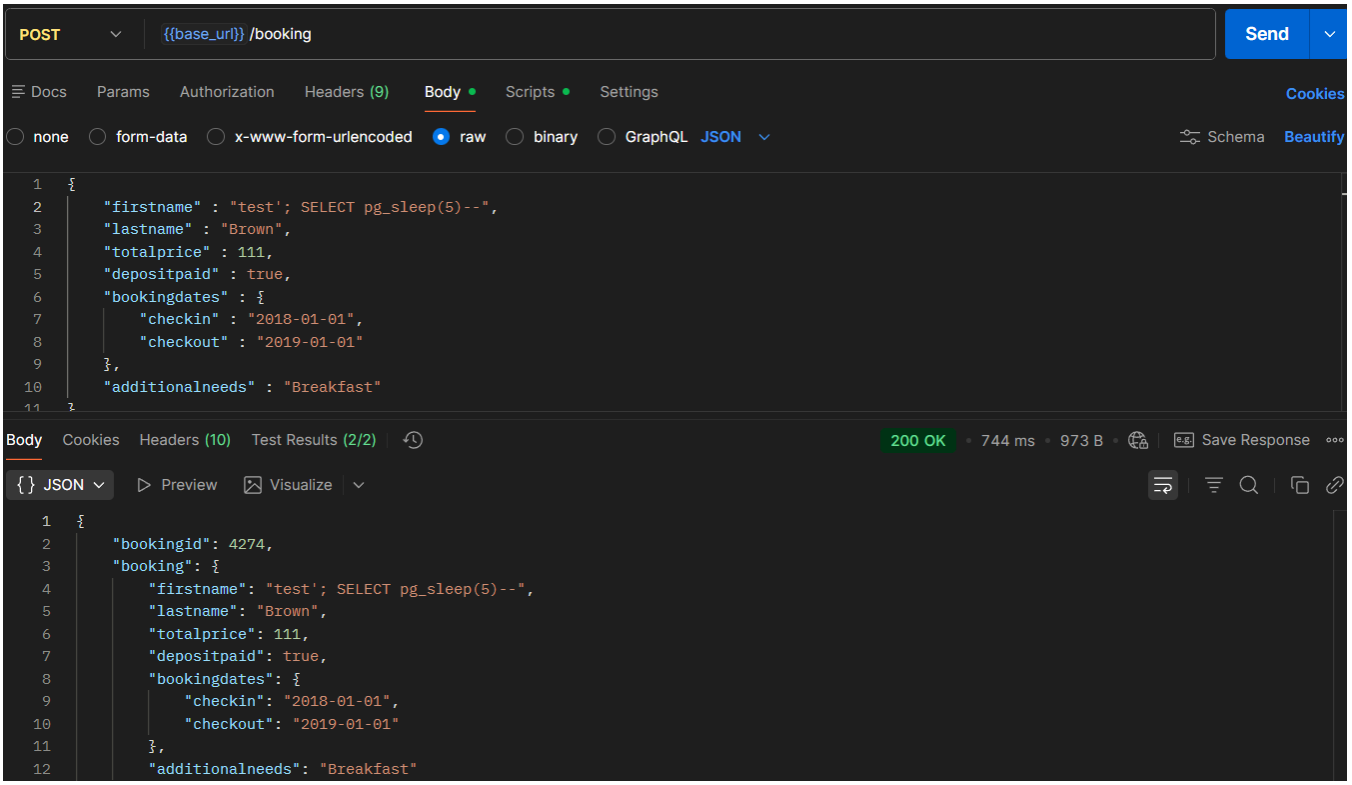
Evidencias de la prueba

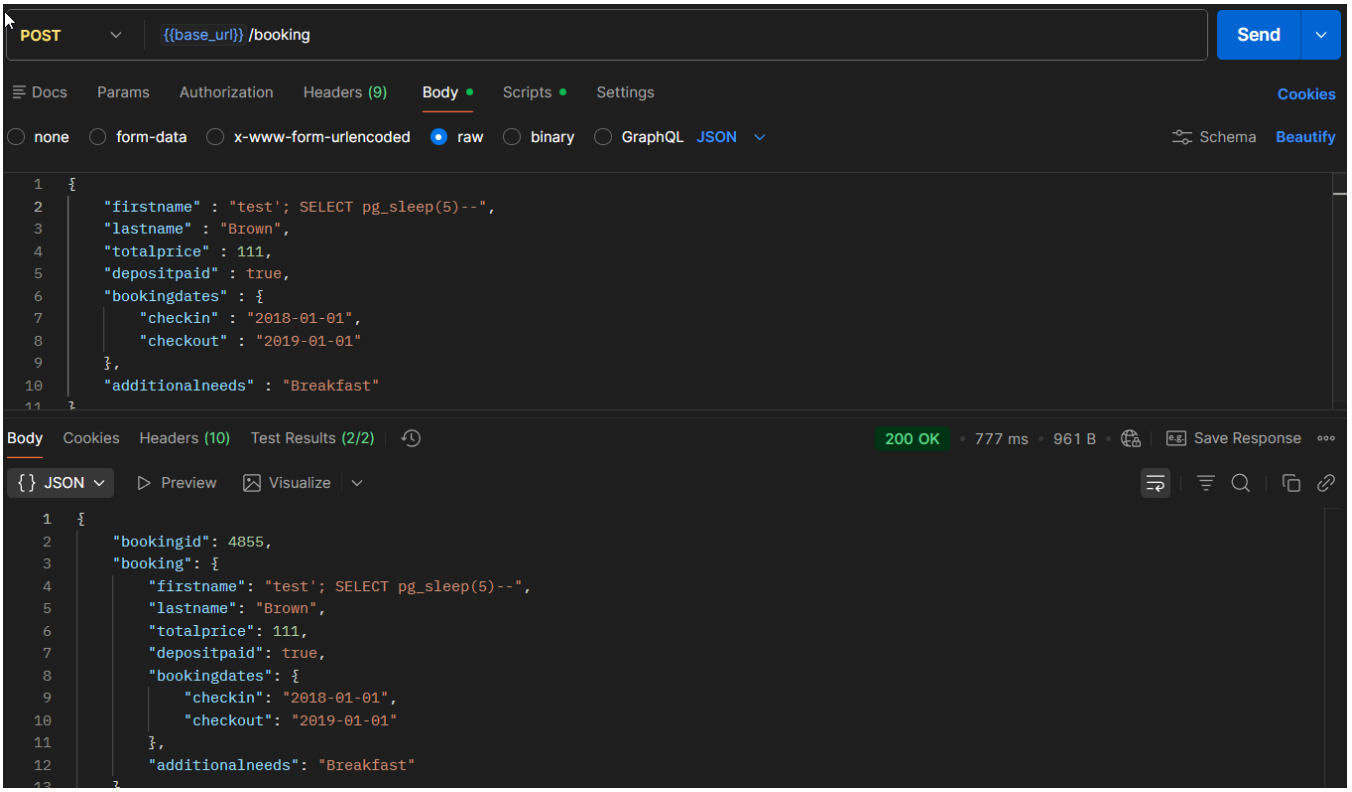
Request legítimo



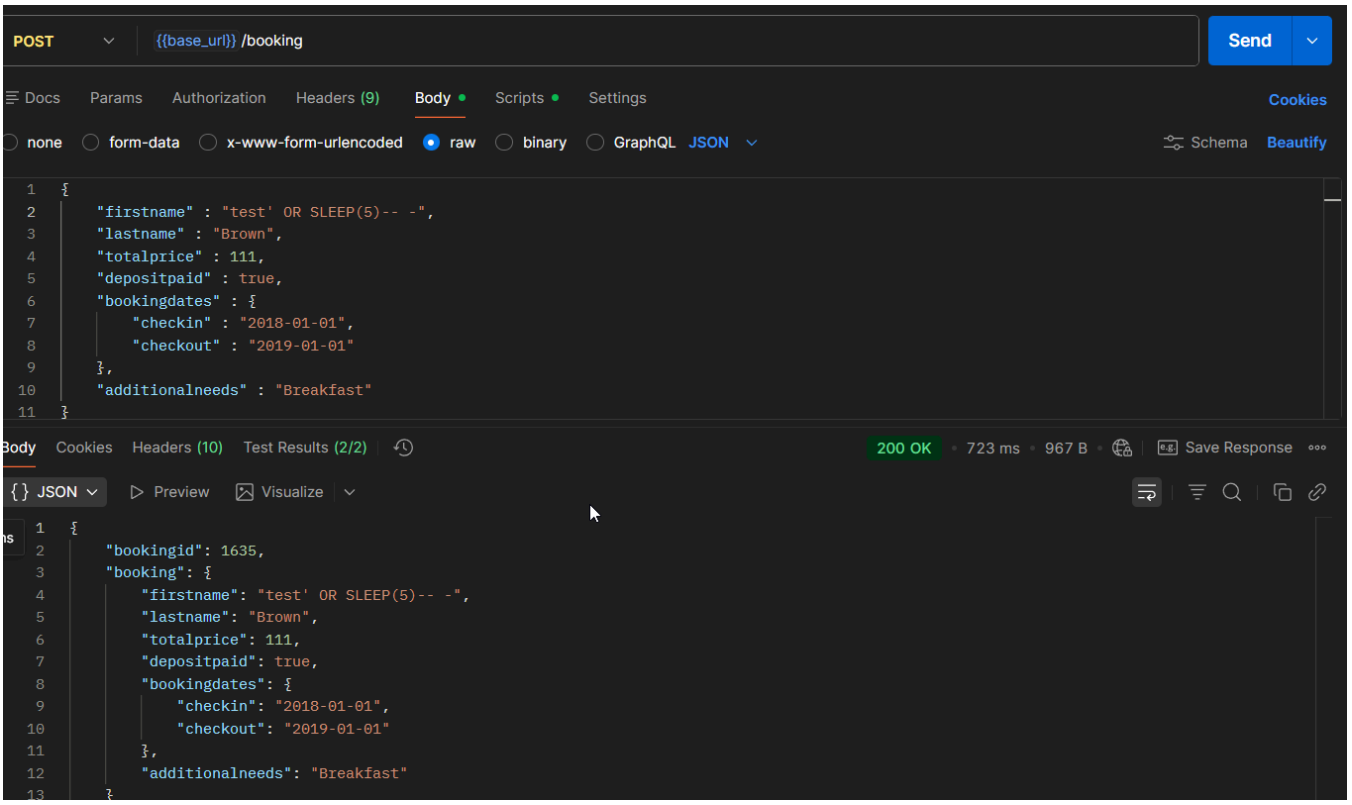
Malicioso PostgreSQL:

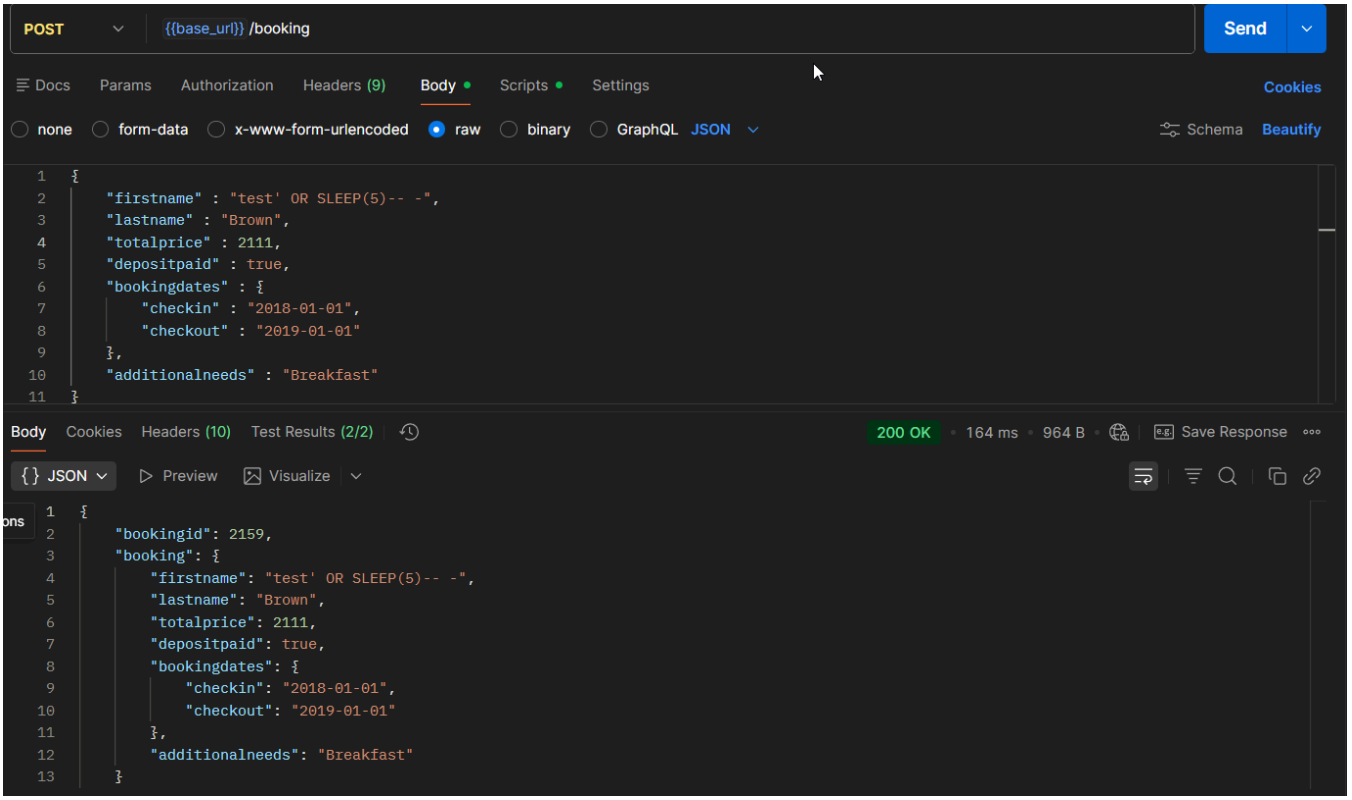


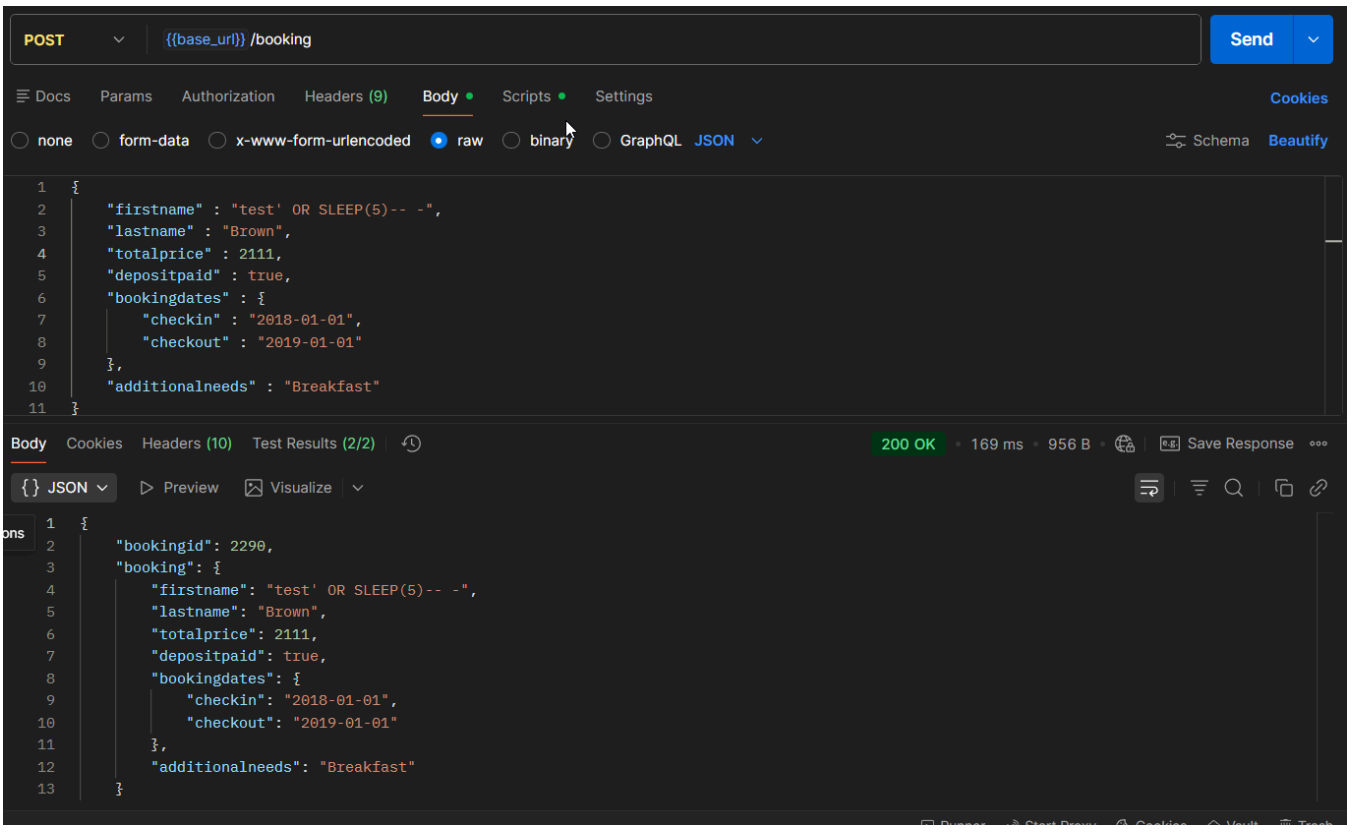




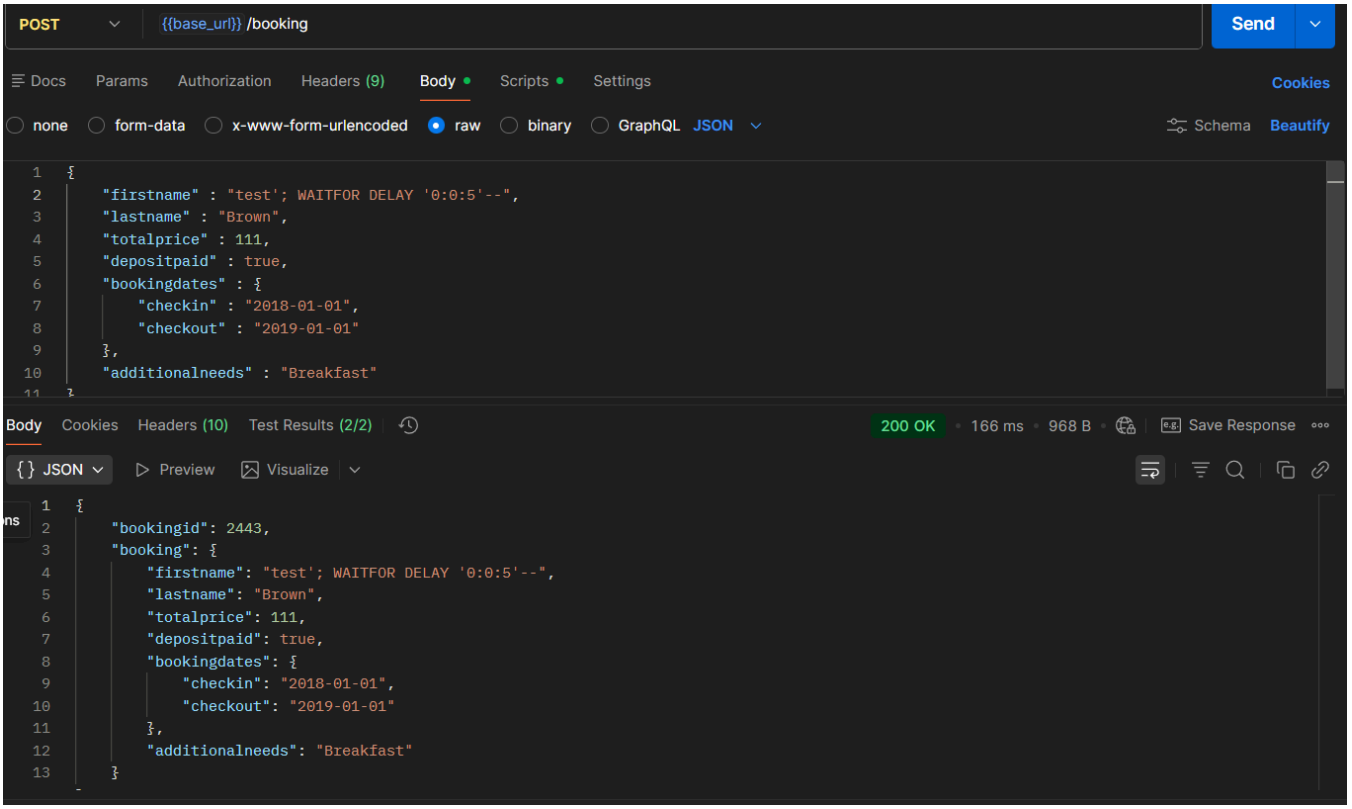
Malicioso MySQL:

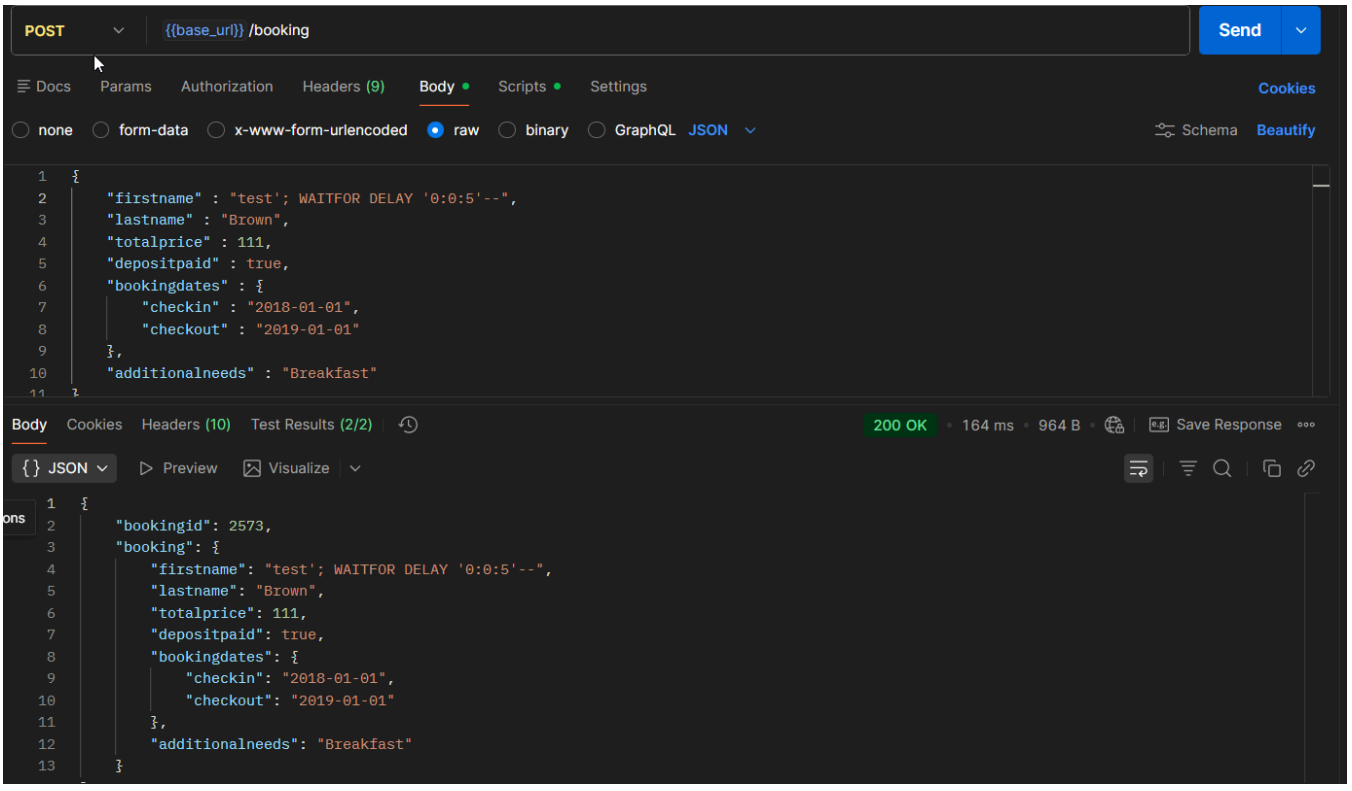


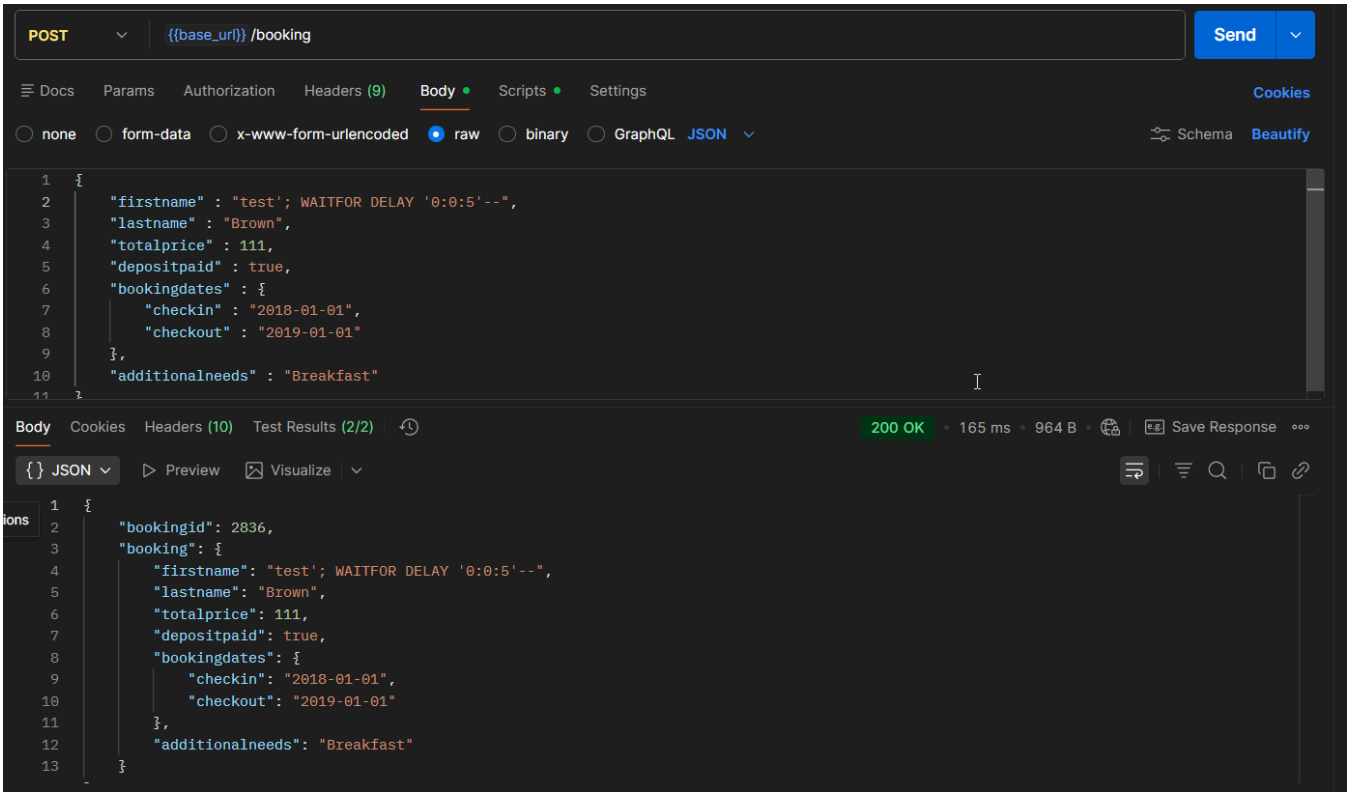

POST
{{base_url}} /booking
Send
Docs Params Authorization Headers (9) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Schema Beautify
{
"firstname" : "test' OR SLEEP(5)-- -",
"lastname" : "Brown",
"totalprice" : 2111,
"depositpaid" : true,
"bookingdates" : {
"checkin" : "2018-01-01",
"checkout" : "2019-01-01"
},
"additionalneeds" : "Breakfast"
}
Body Cookies Headers (10) Test Results (2/2)
200 OK 164 ms 964 B Save Response
{} JSON Preview Visualize
{
"bookingid": 2159,
"booking": {
"firstname": "test' OR SLEEP(5)-- -",
"lastname": "Brown",
"totalprice": 2111,
"depositpaid": true,
"bookingdates": {
"checkin": "2018-01-01",
"checkout": "2019-01-01"
},
"additionalneeds": "Breakfast"
}


POST
{{base_url}} /booking
Send
Docs Params Authorization Headers (9) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Schema Beautify
{
"firstname" : "test' OR SLEEP(5)-- -",
"lastname" : "Brown",
"totalprice" : 2111,
"depositpaid" : true,
"bookingdates" : {
"checkin" : "2018-01-01",
"checkout" : "2019-01-01"
},
"additionalneeds" : "Breakfast"
}
Body Cookies Headers (10) Test Results (2/2)
200 OK 169 ms 956 B Save Response
{} JSON Preview Visualize
{
"bookingid": 2290,
"booking": {
"firstname": "test' OR SLEEP(5)-- -",
"lastname": "Brown",
"totalprice": 2111,
"depositpaid": true,
"bookingdates": {
"checkin": "2018-01-01",
"checkout": "2019-01-01"
},
"additionalneeds": "Breakfast"
}

Malicioso SQL server:







4.2 Vulnerabilidades varias en validación y codificado de Input & Output

Se evaluaron múltiples vectores de inyección (HTML Injection, Script Tag Injection, Stored XSS). Dado que todos los vectores resultaron ejecutables, comparten la misma causa raíz y las mismas recomendaciones de mitigación.

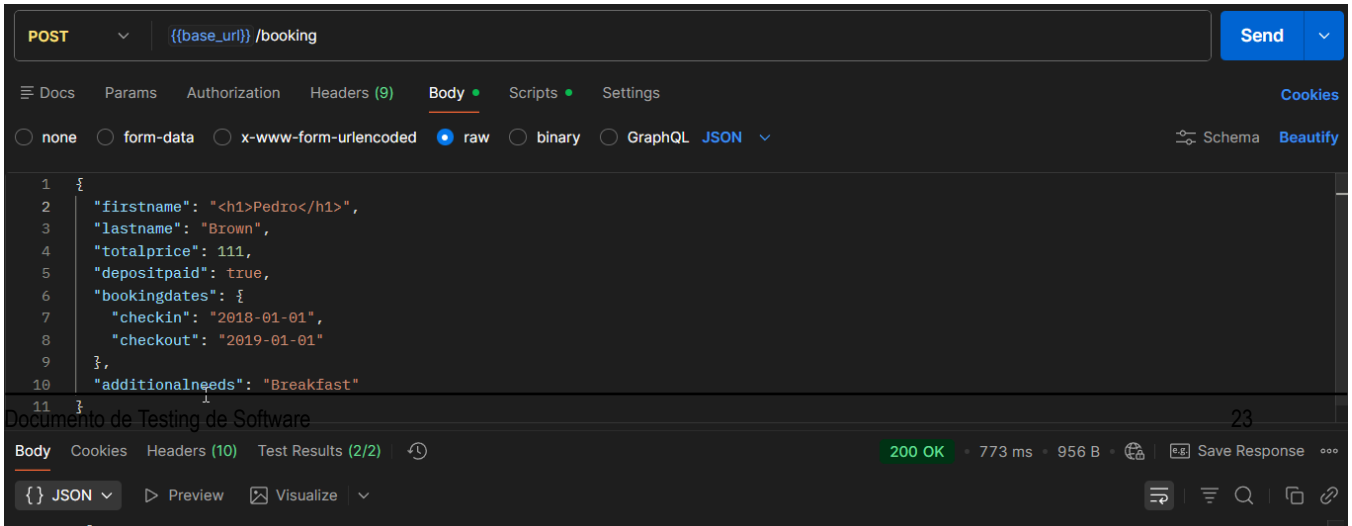
3.2.1 HTML Injection

IDENTIFICADOR	IV-01
DESCRIPCIÓN	Esta prueba verifica si la API valida y sanitiza correctamente contenido HTML enviado en campos controlados por el usuario. El objetivo es determinar si etiquetas HTML pueden ser almacenadas y devueltas sin escapar, lo que podría afectar a cualquier aplicación cliente que renderice estos datos y evidencia una validación de entradas insuficiente.
PRECONDICIONES	Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com API en funcionamiento
DATOS DE ENTRADA	Body de POST Request: {

	"firstname": " Pedro ",
	"lastname": "Test",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2020-01-01",
	"checkout": "2020-01-02"
	},
	"additionalneeds": "Breakfast"
	}
	Request GET:
	https://restful-booker.herokuapp.com/booking/(id de POST)

PASOS A SEGUIR	• Enviar una solicitud POST /booking con el payload indicado. • Verificar el código de estado de la respuesta. • Obtener el bookingid generado. • Ejecutar una solicitud GET /booking/{bookingid}.
RESULTADO ESPERADO	• Status 400 Bad Request, o el campo "<h1>...</h1>" devuelto escapado en los body response. • Ejemplo: <h1>.
RESULTADO OBTENIDO	• Status code casos 200 y body response no escapado.
ESTADO	**FAIL**
SEVERIDAD	**ALTA**
PRIORIDAD	**ALTA**
MITIGACIÓN SUGERIDA	La vulnerabilidad debe ser mitigada implementando sanitización de entrada del lado del servidor y codificado seguro de salida, validacion estricta de modelos y testeo automatizado de seguridad. Además puede implementarse, como capa de seguridad adicional, configuración WAF de bloqueo de patrones típicos de XSS.

Evidencias de la prueba



3.2.2 Stored XSS y Tag Injection

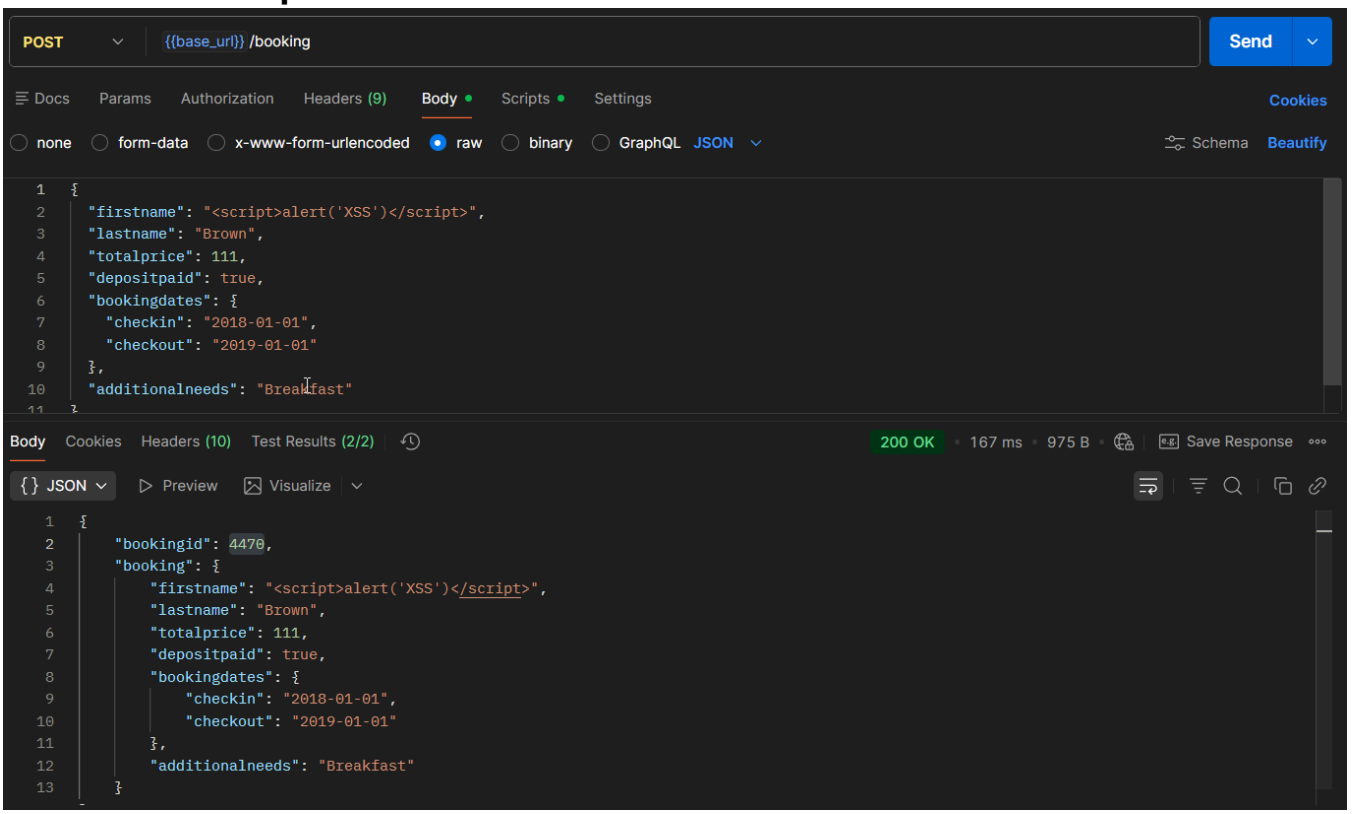
IDENTIFICADOR	IV-02
DESCRIPCIÓN	Esta prueba verifica si el endpoint POST /booking permite almacenar código HTML/JavaScript malicioso (Stored XSS), lo cual podría ser ejecutado posteriormente en una interfaz cliente o panel administrativo que consuma los datos devueltos por la API.
PRECONDICIONES	Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com API en funcionamiento

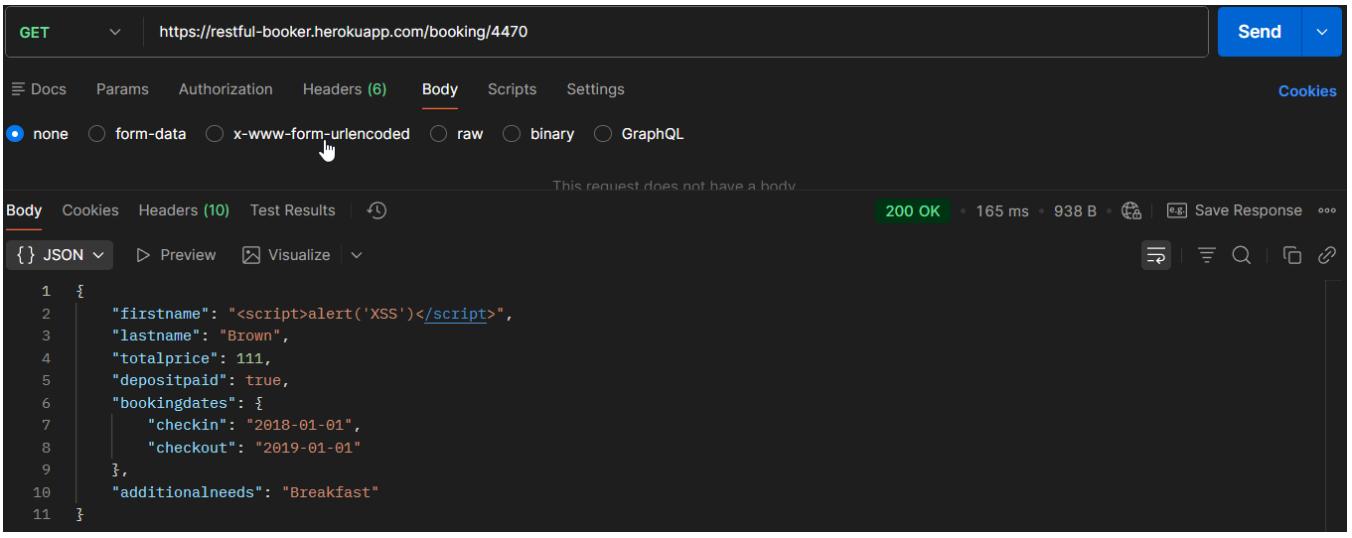
DATOS DE ENTRADA	Body de POST Request:
	{
	"firstname": "<script>alert('XSS')</script>",
	"lastname": "Brown",
	"totalprice": 111,
	"depositpaid": true,
	"bookingdates": {
	"checkin": "2018-01-01",
	"checkout": "2019-01-01"
	},
	"additionalneeds": "Breakfast"
	}

PASOS A SEGUIR	● Enviar request POST URL: {{base_url}}/booking con el payload malicioso anterior. ● Registrar status code y response body ● Enviar request GET URL: {{base_url}}/booking/(id del POST)
RESULTADO ESPERADO	● Status 400 Bad Request, o el campo “<script>...</script>” devuelto escapado en los body response. ● Ejemplo: <script>alert('XSS')</script>
RESULTADO OBTENIDO	● Status code casos 200 y body response no escapado.
ESTADO	**Fail**
SEVERIDAD	**ALTA**
PRIORIDAD	**ALTA**
MITIGACIÓN SUGERIDA	Misma raíz de problema que el defecto de HTMLi y, por ende, mismas sugerencias: La vulnerabilidad debe ser mitigada implementando sanitización de entrada del lado del servidor y codificado

	seguro de salida, validacion estricta de modelos y testeo automatizado de seguridad. Además puede implementarse, como capa de seguridad adicional, configuración WAF de bloqueo de patrones típicos de XSS.

Evidencia de la prueba:





4.3 Access Control

Este bloque verifica que la API proteja correctamente los recursos sensibles y no permita operaciones críticas sin autenticación válida.

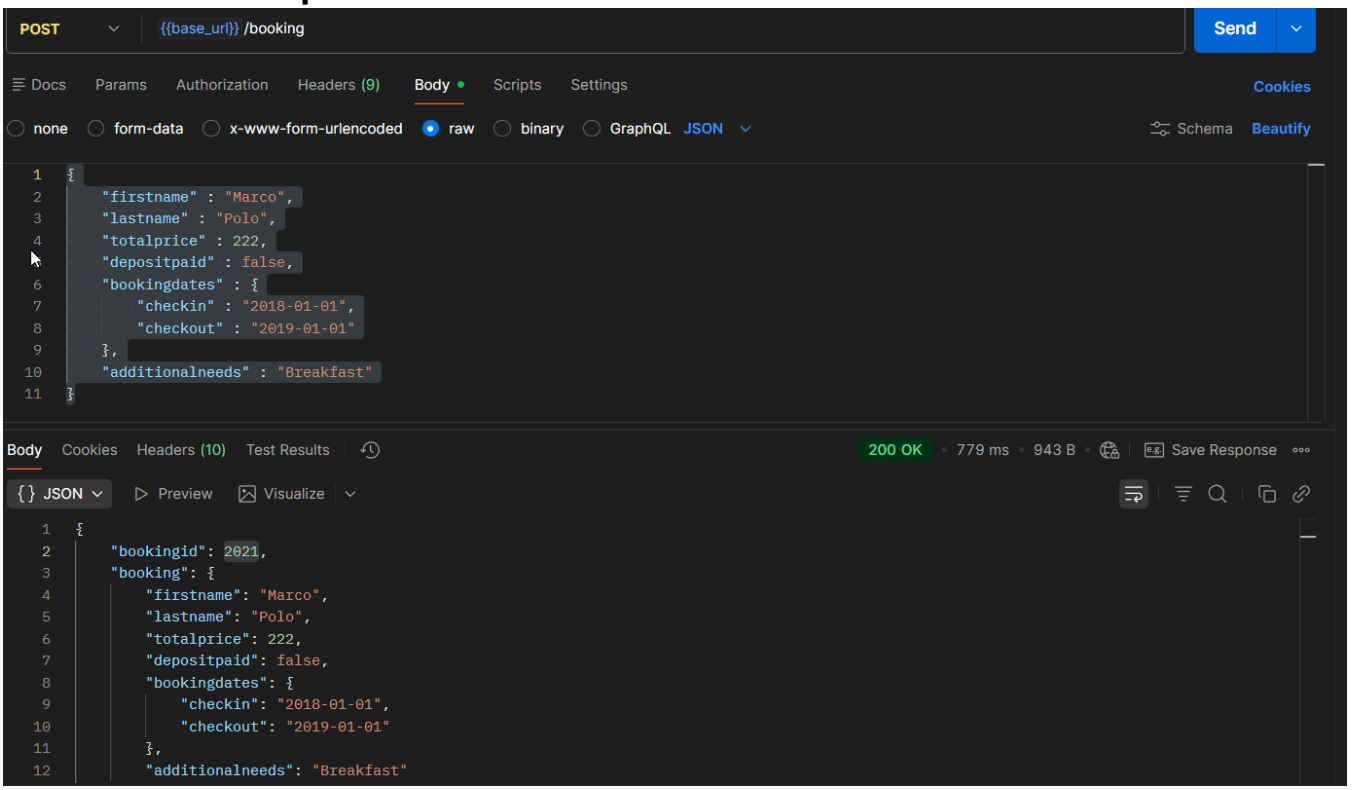
3.3.1 Acceso a recurso protegido sin token

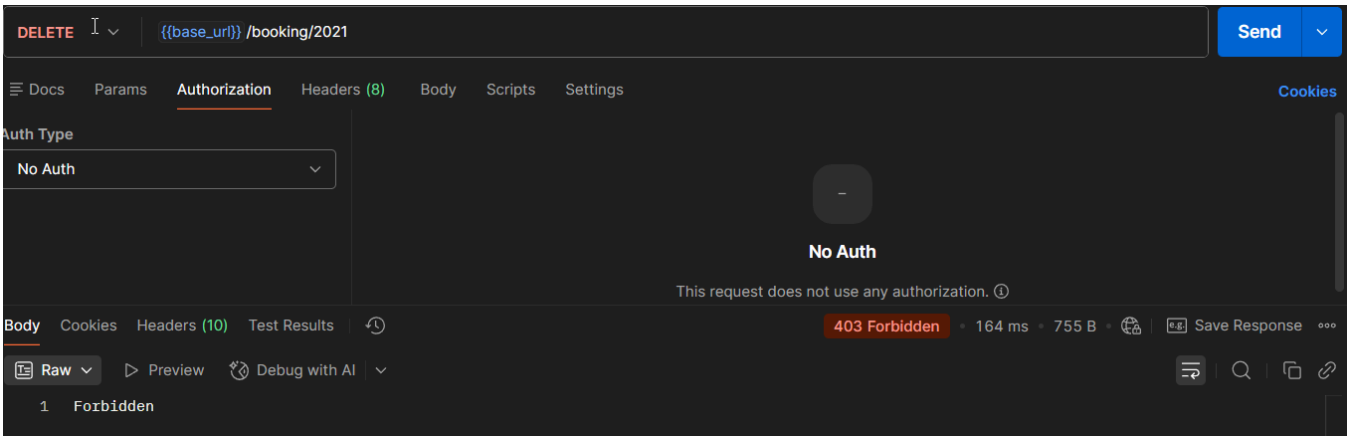
IDENTIFICADOR	A-01
DESCRIPCIÓN	Esta prueba valida que la API bloquee el acceso a endpoints protegidos cuando la solicitud no incluye un token de autenticación válido. El objetivo es verificar que operaciones sensibles (modificación o eliminación de recursos) no puedan ejecutarse por usuarios no autenticados.
PRECONDICIONES	Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com API en funcionamiento

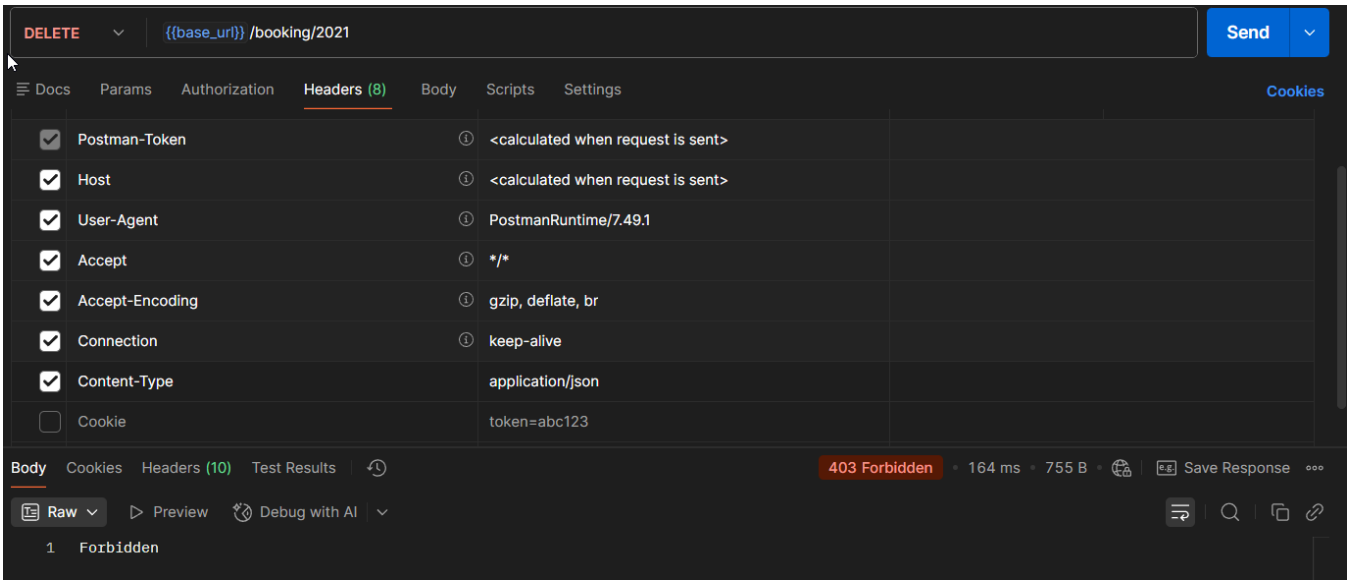
SET UP DE DATOS	Body de POST:
	{
	"firstname" : "Marco",
	"lastname" : "Polo",
	"totalprice" : 222,
	"depositpaid" : false,
	"bookingdates" : {
	"checkin" : "2018-01-01",
	"checkout" : "2019-01-01"
	},
	"additionalneeds" : "Breakfast"
	}

PASOS A SEGUIR	● Enviar request POST URL: {{base_url}}/booking ● Enviar request DELETE URL: {{base_url}}/booking/(id del POST) sin token ni cookies de autorización
RESULTADO ESPERADO	● Status code 401 o 403
RESULTADO OBTENIDO	● Status code 401 o 403
ESTADO	**Pass**

Evidencias de la prueba:







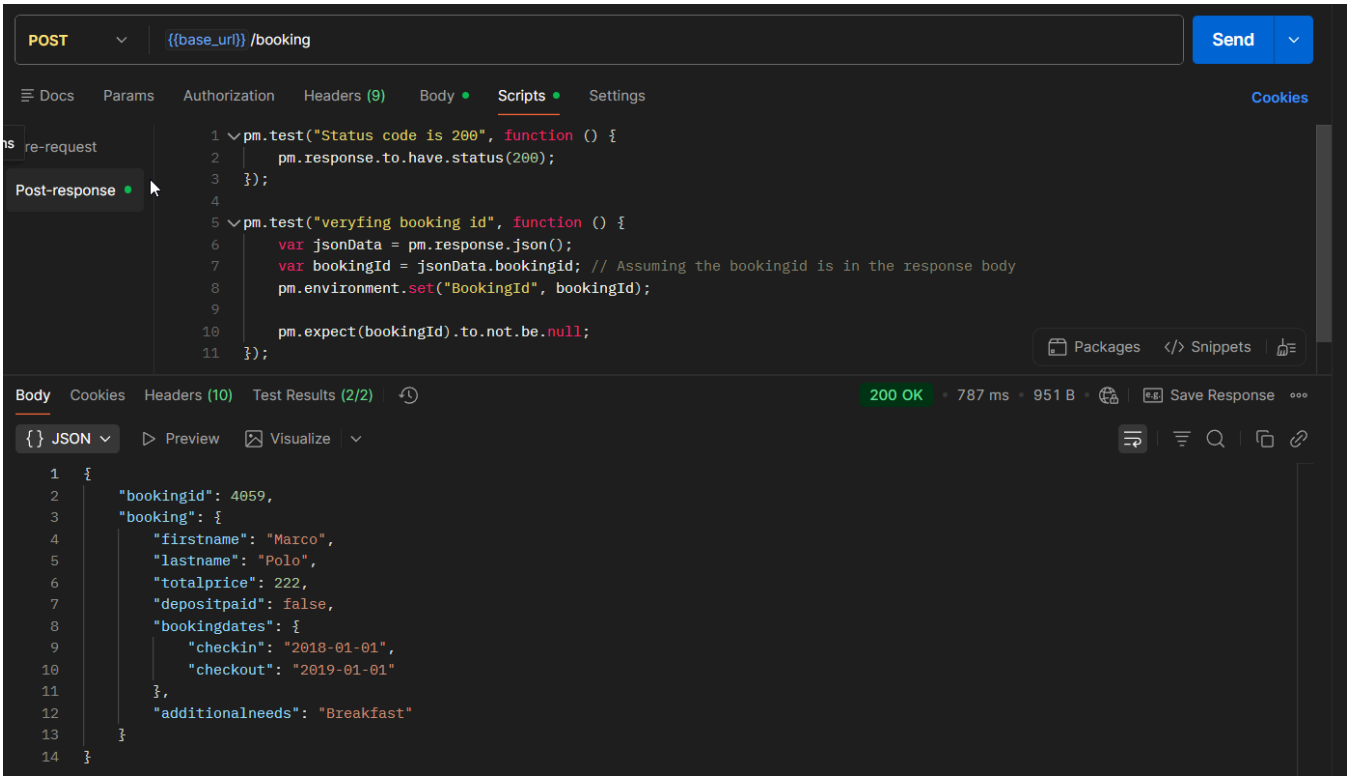
3.3.2 Acceso a recurso protegido con token inválido

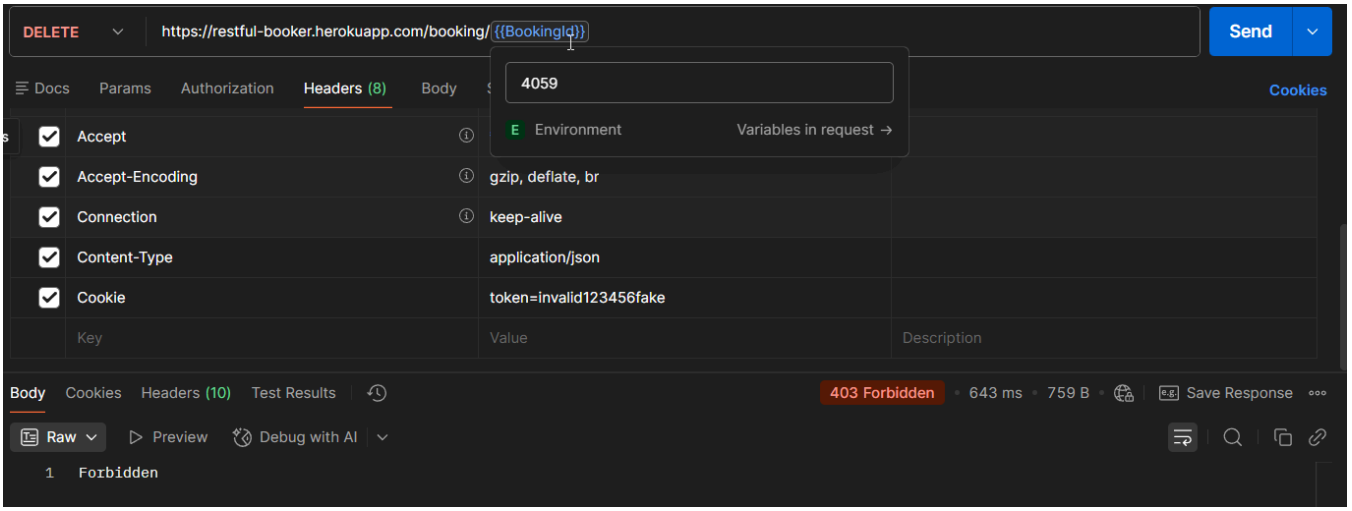
IDENTIFICADOR	A-02
DESCRIPCIÓN	Esta prueba valida que la API bloquee el acceso a endpoints protegidos cuando la solicitud incluye tokens de autorización inválidos. El objetivo es verificar que operaciones sensibles (modificación o eliminación de recursos) no puedan ejecutarse por usuarios no autenticados.
PRECONDICIONES	Postman configurado con {{base_url}} = https://restful-booker.herokuapp.com API en funcionamiento

SET UP DE DATOS	Body de POST:
	{
	"firstname" : "Marco",
	"lastname" : "Polo",
	"totalprice" : 222,
	"depositpaid" : false,
	"bookingdates" : {
	"checkin" : "2018-01-01",
	"checkout" : "2019-01-01"
	},
	"additionalneeds" : "Breakfast"
	}
	Cookie en DELETE request: token=invalid123456fake

PASOS A SEGUIR	● Enviar request POST URL: {{base_url}}/booking ● Enviar request DELETE URL: {{base_url}}/booking/(id del POST) con cookie token inválido
RESULTADO ESPERADO	● Status code 401 o 403
RESULTADO OBTENIDO	● Status code 401 o 403
ESTADO	**Pass**

Evidencias de la prueba:





Conclusión Ejecutiva

El presente documento detalla el resultado de las pruebas de testing funcional y de seguridad realizadas sobre la API Restful Booker, con foco en vulnerabilidades comunes de aplicaciones web (OWASP Top 10), particularmente inyecciones SQL (SQLi), validación y codificación de entradas/salidas (Input & Output Handling) y control de accesos.

Durante la ejecución de los casos de prueba relacionados con SQL Injection, no se detectaron vulnerabilidades explotables. En todos los escenarios evaluados (boolean-based, error-based y time-based), la API respondió de manera consistente, sin exponer errores internos, sin variaciones significativas en las respuestas y sin retrasos anómalos en los tiempos de ejecución. Estos resultados indican que, al menos desde el punto de vista observado, el backend no es vulnerable a inyecciones SQL inferenciales ni directas, o bien no ejecuta consultas SQL dinámicas sobre los campos evaluados.

En cuanto al control de accesos, las pruebas confirmaron que los endpoints críticos se encuentran correctamente protegidos. Los intentos de eliminación de recursos sin autenticación o con tokens inválidos fueron bloqueados adecuadamente, devolviendo códigos de estado HTTP coherentes (401/403). Esto demuestra una correcta implementación de los mecanismos básicos de autorización para operaciones sensibles.

No obstante, se identificaron vulnerabilidades críticas en la validación y sanitización de datos de entrada, específicamente relacionadas con HTML Injection y Stored Cross-Site Scripting (XSS). La API permite almacenar y devolver contenido HTML y JavaScript sin ningún tipo de escape o validación, lo cual representa un riesgo significativo para cualquier aplicación cliente que consuma estos datos y los renderice en un contexto web. Estas vulnerabilidades no afectan directamente a la API en aislamiento, pero sí introducen un vector de ataque severo en escenarios reales de integración con frontends o paneles administrativos.

En conclusión, si bien la API presenta un buen comportamiento frente a ataques de inyección SQL y controles de acceso, requiere mejoras urgentes en la validación y codificación de entradas y salidas. Se recomienda implementar sanitización del lado del servidor, validaciones estrictas de modelos de datos, codificado seguro de salida y pruebas automatizadas de seguridad como parte del pipeline de desarrollo. Estas acciones permitirán reducir significativamente la superficie de ataque y alinear el sistema con buenas prácticas de seguridad aplicables a entornos productivos.