Final Project 2025

Keamanan Jaringan

GROUP 7

SQL INJECTION & COMMAND INJECTION

Wilman Saragih Sitio 2306161776

Musyaffa Iman Supriadi 2306208464

Latar Belakang

Memberikan contoh nyata mengenai teknik dan pola serangan pada keamanan jaringan sekaligus mendemonstrasikan cara kerja dan langkah remediasi melalui proses simulasi *penetration testing*.

Tujuan

1. Mendemonstrasikan eksploitasi SQL Injection & Command Injection.
2. Mengevaluasi dampak kerusakan sistem.
3. Menerapkan dan memverifikasi langkah remediasi (perbaikan).

Ruang Linkgup

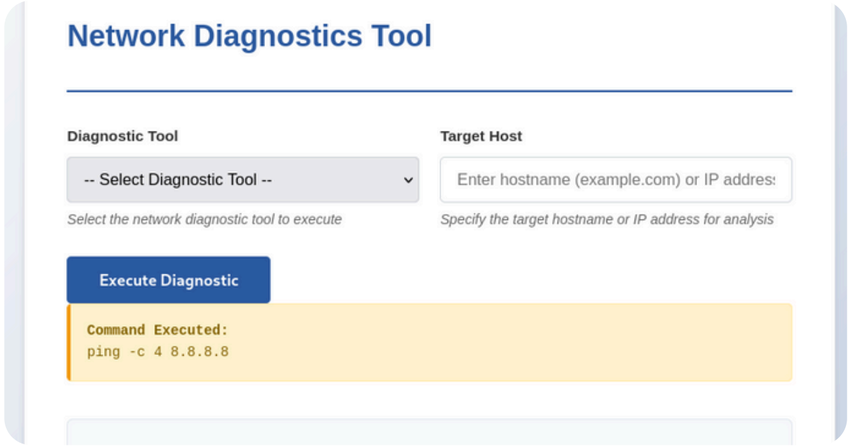
- **Target**: Aplikasi Web Self-Hosted (TechStore & Network Tools).
- **Lingkungan**: Virtualisasi (Kali Linux & Server Target) yang terisolasi.

Metodologi

1. Reconnaissance: Identifikasi port & layanan.
2. Enumeration: Memetakan titik masuk (input form).
3. Exploitation: Eksekusi payload serangan.
4. Remediation: Patching celah keamanan.

Tech Stack **VM** **OS**

Target Creation
Command Injection



Command Injection

01 Command Injection (Linux OS Command).

02 Penyebab: Kegagalan aplikasi memvalidasi input sebelum diproses oleh system shell.

03 Dampak: Eksekusi perintah (RCE), kebocoran data, pengambilalihan server.

Deskripsi Aplikasi

Uji konektivitas dan analisis jaringan standar, dengan Ping, NSLookup, Traceroute, dan WHOIS. Bekerja dengan menerima input alamat IP dari pengguna, lalu mengeksekusinya secara langsung ke system shell

Enumeration
Command Injection

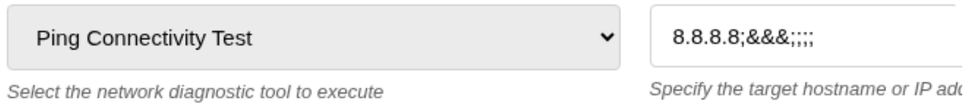
Port Scanning (Nmap)

- Menemukan Port 80 (HTTP) Open.
- Service: Apache httpd 2.4.65 (Debian).

```
┌──(wilman㉿wilman)-[~]
└─$ nmap -sV -p- 192.168.1.72
Starting Nmap 7.95 ( https://nmap.org ) at 2025-1
Nmap scan report for wilman.domain.name (192.168.
Host is up (0.000012s latency).
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.65 ((Debian

Service detection performed. Please report any in
Nmap done: 1 IP address (1 host up) scanned in 8.
```

Analisis Burp Suite

- Intercept request saat melakukan ping normal ke 8.8.8.8.
- Aplikasi mengirim data via POST. Karakter spesial seperti ;, &, |, / otomatis diubah menjadi URL Encoding %3B, %26, %7C, %2F.




```
1  POST /network-tools/ HTTP/1.1
2  Host: 192.168.1.72
3  Content-Length: 48
4  Cache-Control: max-age=0
5  Accept-Language: en-US,en;q=0.9
6  Origin: http://192.168.1.72
7  Content-Type: application/x-www-form-urlencoded
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
   Chrome/139.0.0.0 Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
   png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://192.168.1.72/network-tools/
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 tool=ping&target=8.8.8.8%3B%26%26%26%3B%3B%3B%3B
```

Exploitation Command Injection

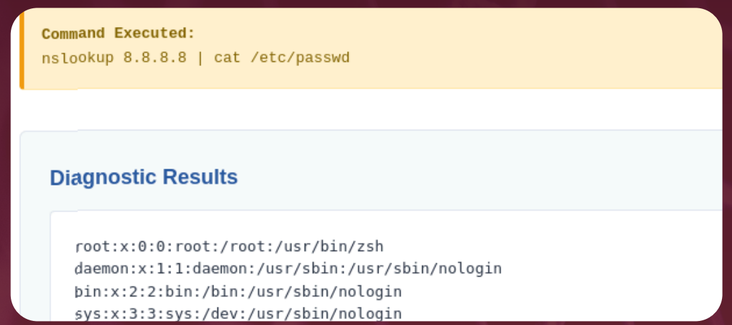
Teknik Command Chaining (menggabungkan perintah).

Operator & (Ampersand)

- Payload: 8.8.8.8 & ls
- Menampilkan daftar file direktori.

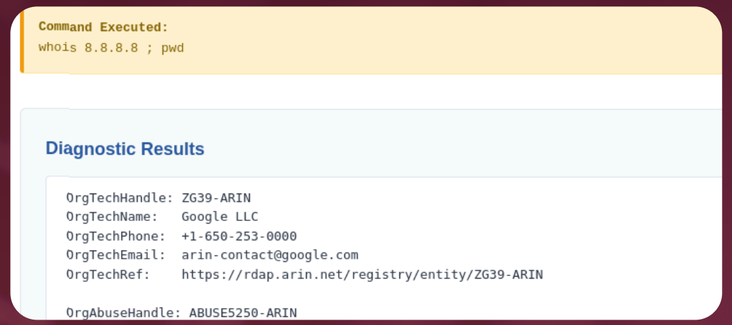
Operator | (Pipe)

- Payload: 8.8.8.8 | cat /etc/passwd
- Membaca file sensitif sistem.



Operator ; (Semicolon)

- Payload: 8.8.8.8 ; pwd
- Mengetahui path direktori kerja server.



Exploitation Command Injection

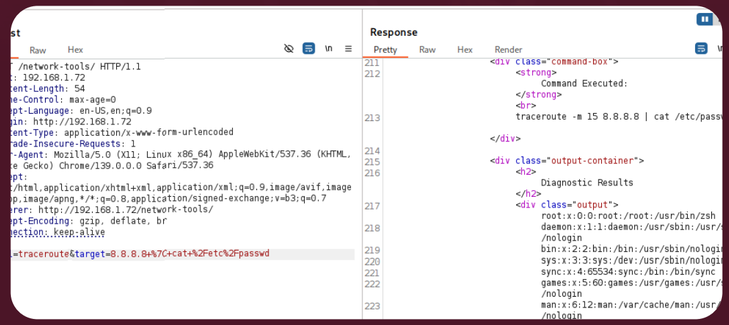
Menggunakan Burp Suite

→ Operator & (Ampersand)

- Payload: tool=ping&target=8.8.8.8+%26+ls
- Menampilkan daftar file direktori.

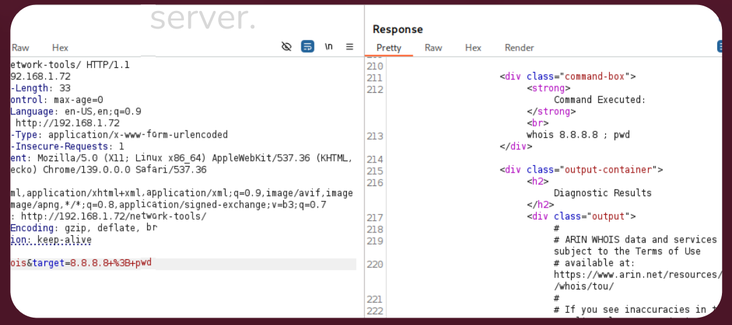
→ Operator | (Pipe)

- Payload: tool=traceroute&target=8.8.8.8+%7C+cat+%2Fetc%2Fpasswd
- Membaca file sensitif sistem.



→ Operator ; (Semicolon)

- Payload: tool=whois&target=8.8.8.8+%3B+pwd
- Mengetahui path direktori kerja server.



01

Root Cause

```
$command = "ping -c 4 {$target}";
$command_executed = $command;
```

Masalah:

- Variabel $target diambil mentah dari input user.
- Tidak ada validasi atau sanitasi.
- Fungsi shell_exec mengeksekusi apa saja yang diberikan.

02

Remediasi

Penerapan Input Sanitization menggunakan escapeshellarg().

- Kode Secure:

```
$safe_target = escapeshellarg($target);
```

- Membungkus input dengan single quotes (').
- Input 8.8.8.8 ; ls berubah menjadi string '8.8.8.8 ; ls'.
- Sistem menganggapnya sebagai satu argumen nama host, bukan *command* terpisah.

Remediation
Command Injection

```
$tool = $_POST['tool'] ?? '';
$target = $_POST['target'] ?? '';

if (!empty($tool) && !empty($target)) {

    $safe_target = escapeshellarg($target);
```

Selain itu juga diharuskan menghilangkan nested shell:

- Kode Secure:

```
$command_executed = $command;
$output = shell_exec($command);
```

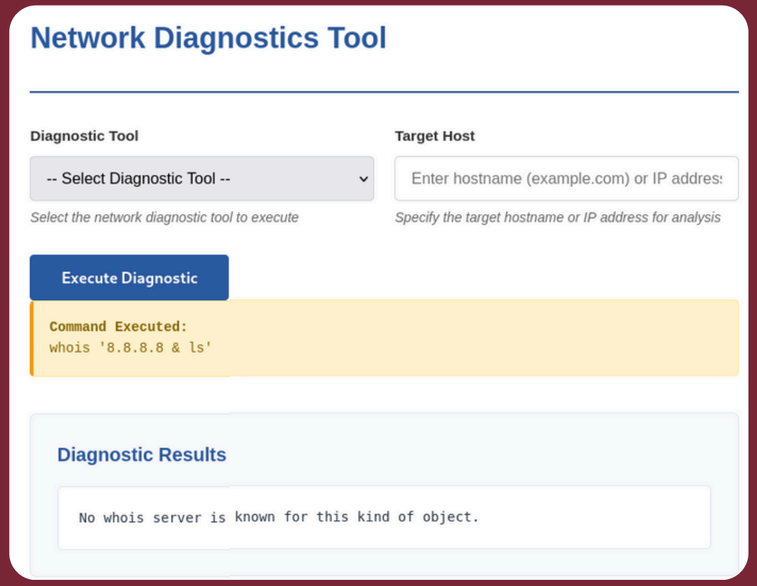
- Mengeliminasi risiko manipulasi.
- Menjalankan perintah yang sudah disanitasi secara langsung.
- Mengurangi kompleksitas kode untuk meminimalisir celah keamanan potensial.

Proof **Command Injection**

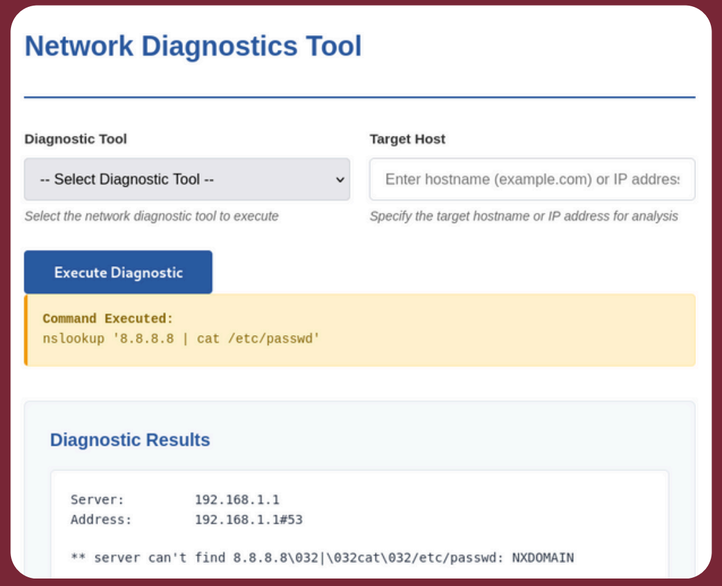
Hasil:

- Serangan Gagal.
- Server tidak memberikan respon dari hasil eksekusi shell.
- Perintah hasil chaining tidak dieksekusi.

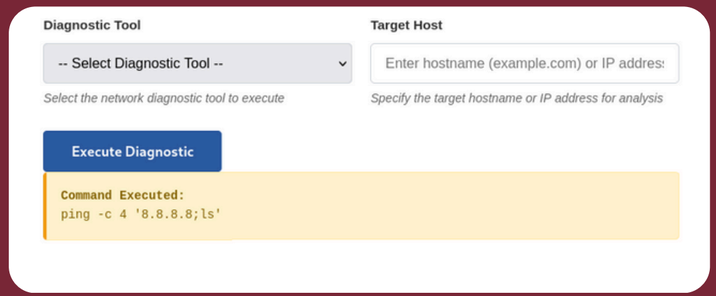
Ampersand (&)



Pipe (|)



Semicolon (;)



FInal Project 2025

Keamanan Jaringan

SQL Injection

SQL Injection (SQLi) adalah jenis serangan injeksi yang memungkinkan attacker memasukkan query SQL jahat melalui input data ke dalam query yang dibuat aplikasi.

Aplikasi web TechStore ini sengaja dibuat rentan untuk tujuan pengujian dan berhasil mengidentifikasi 3 kerentanan SQL Injection Kritis pada endpoint Login, Search, dan User Lookup yang dapat menyebabkan Authentication Bypass dan Data Exfiltration.

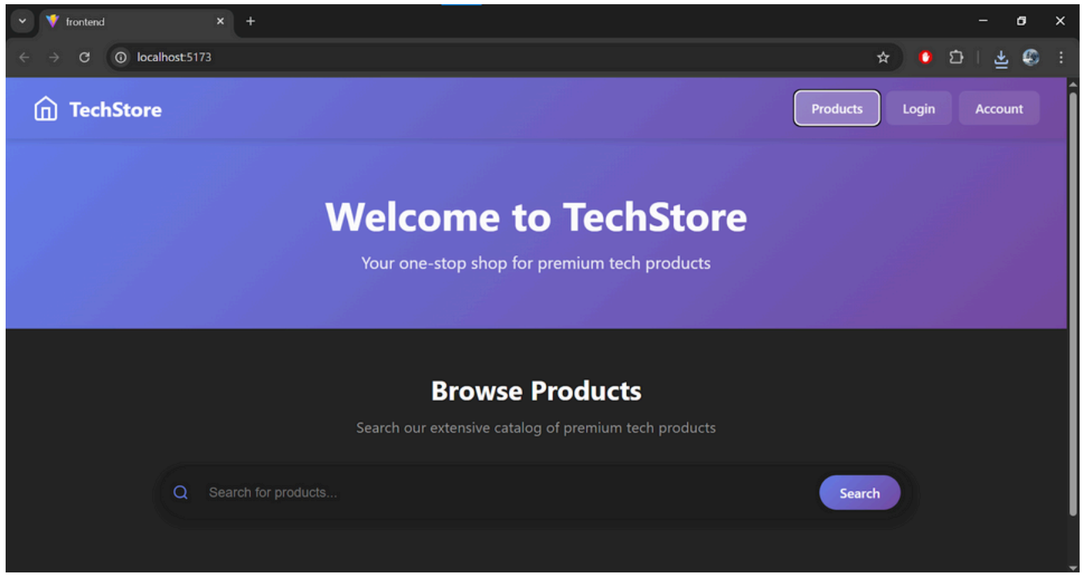
Tech Stack

React JS

SQLite

Express

Target Creation



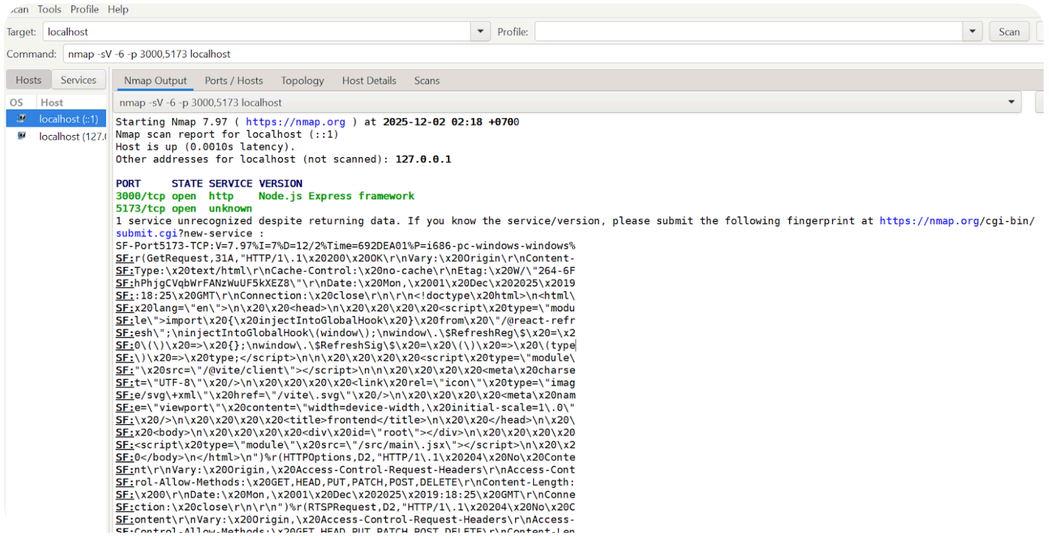
Deskripsi Aplikasi

Aplikasi web e-commerce sederhana yang memfasilitasi otentikasi user dan pencarian data produk. Sistem dirancang rentan dengan cara memproses input mentah dari parameter HTTP langsung ke dalam struktur query SQL, memungkinkan manipulasi logika database melalui injeksi kode.

Enumeration
SQL Injection

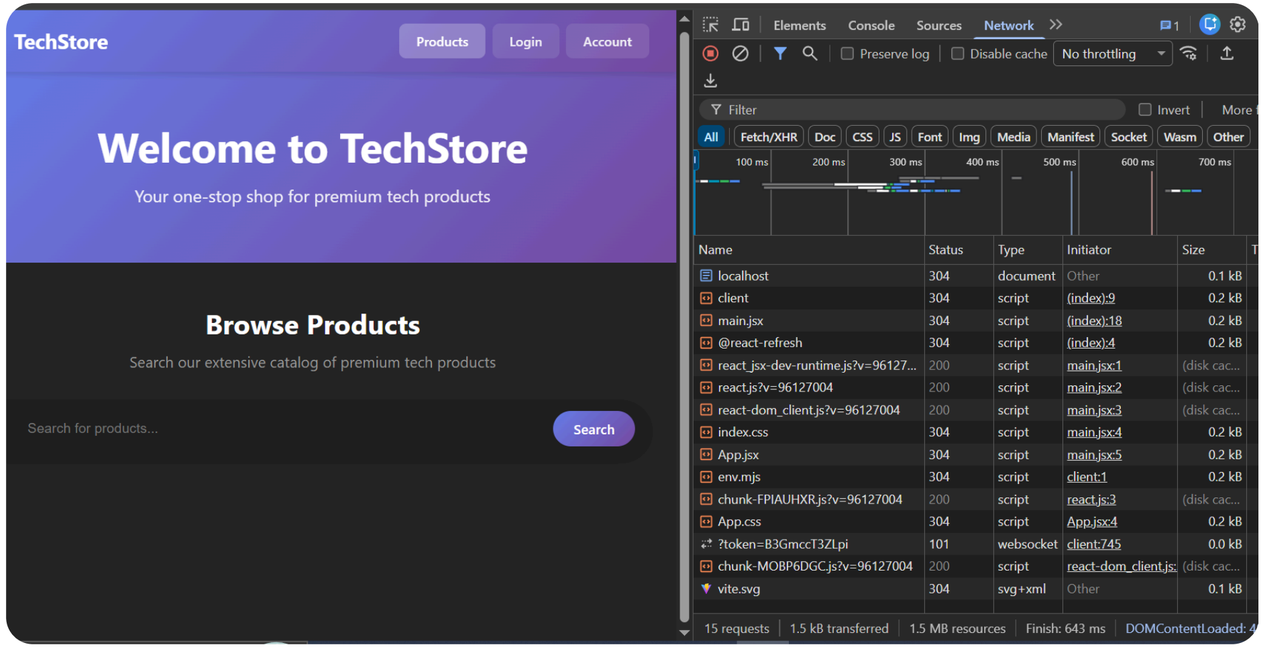
Port Scanning (Nmap)

- Port 3000/tcp: Node.js Express framework (Backend API).
- Port 5173/tcp: Vite dev server (Frontend).

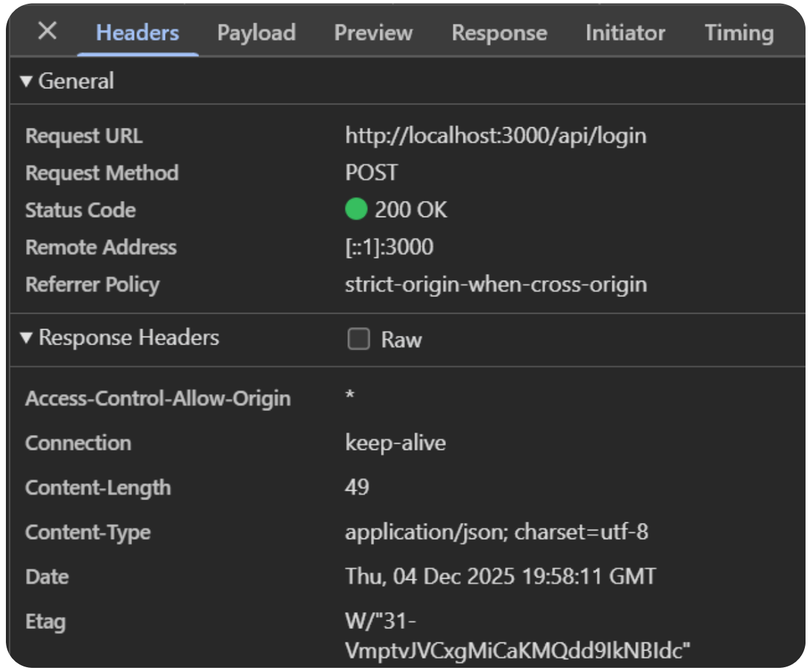


Technology Fingerprinting

- Website menggunakan React sebagai Frontend Framework.
- Website menggunakan Vite sebagai Development server.

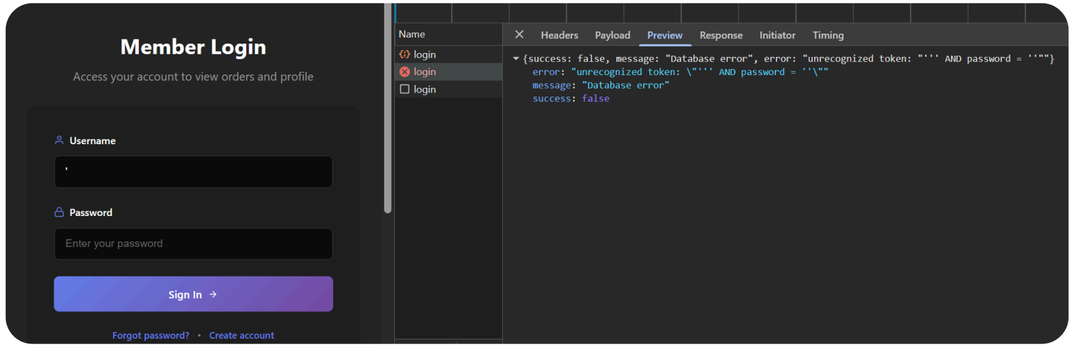


Analyze HTTP Response Headers



- Backend framework menggunakan Express yang secara implisit mengatakan backend berjalan di atas runtime Node.js.
- Response berbentuk format JSON.
- Menggunakan CORS.

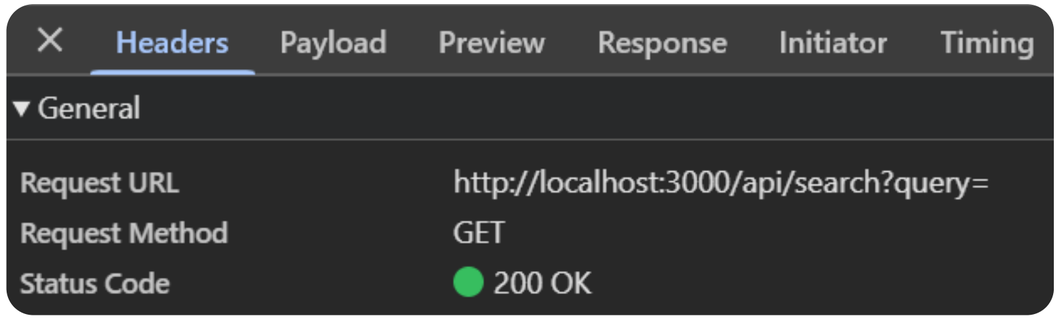
Test Error Messages for Database Info



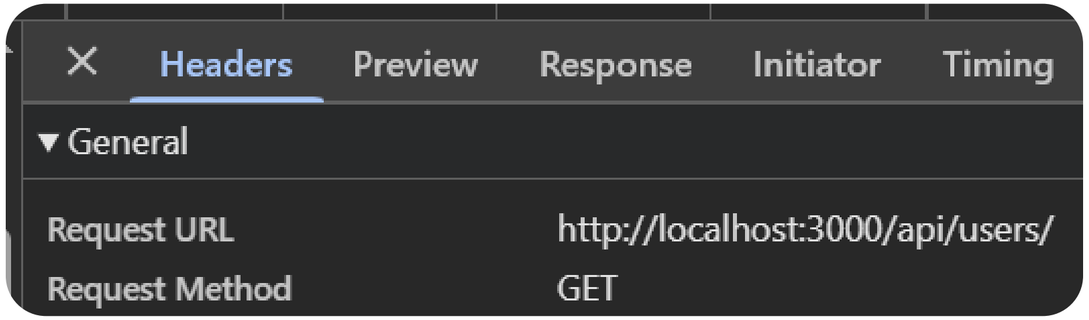
- Pesan error SQL terekspos.
- Struktur query terekspos.
- Pola pesan error mengindikasikan database SQLite.
- Tidak terdeteksi adanya sanitasi input.
- Tidak ada sanitasi pesan error.

Endpoint Dicovery

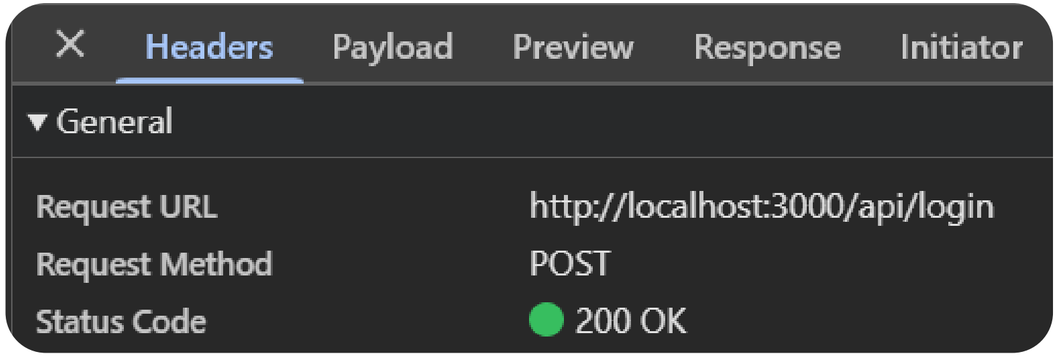
- /api/search



- /api/users



- /api/login

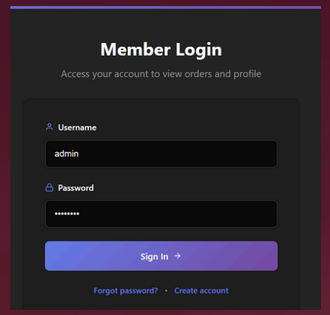


Exploitation SQL Injection

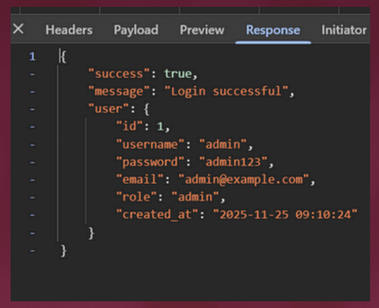
Endpoint Testing

Login Biasa

- Username: admin
- Password: admin123



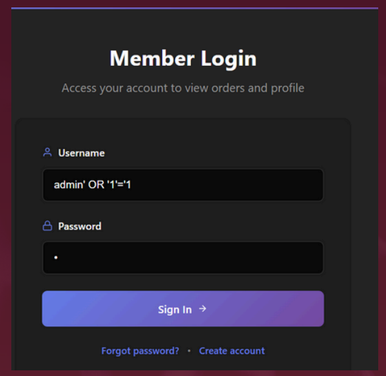
Output:



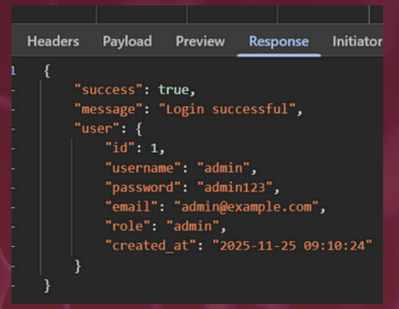
Terlihat detail akun

Authentication Bypass

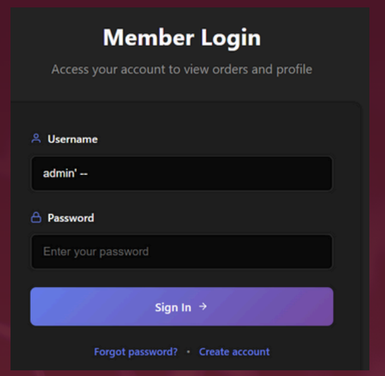
- Username: admin' OR '1'='1
- Password: (bebas)



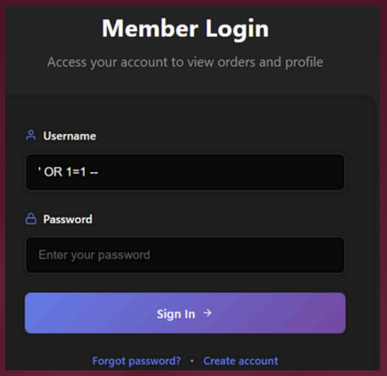
Output:



- Username: admin' --
- Password: (kosong/bebas)



- Username: ' OR 1=1 --
- Password: (bebas)



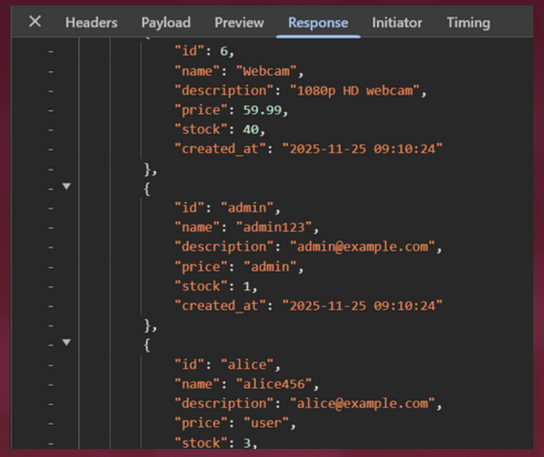
Dapat akses admin tanpa mengetahui password

Exploitation SQL Injection

Endpoint Testing

Data Exfiltration

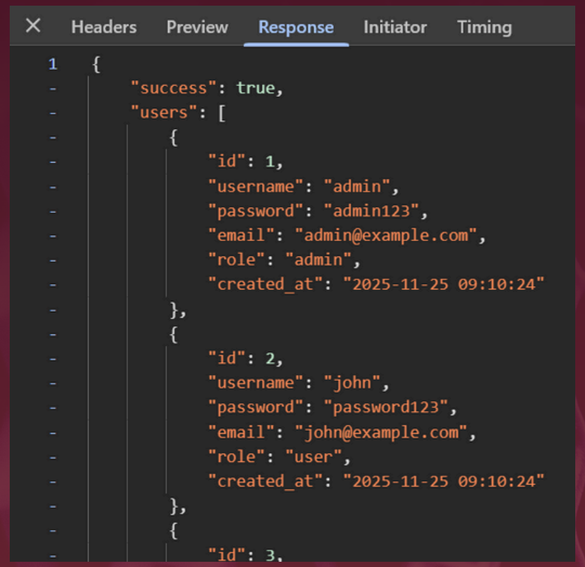
- Query: ' UNION SELECT username, password, email, role, id, created_at FROM users --



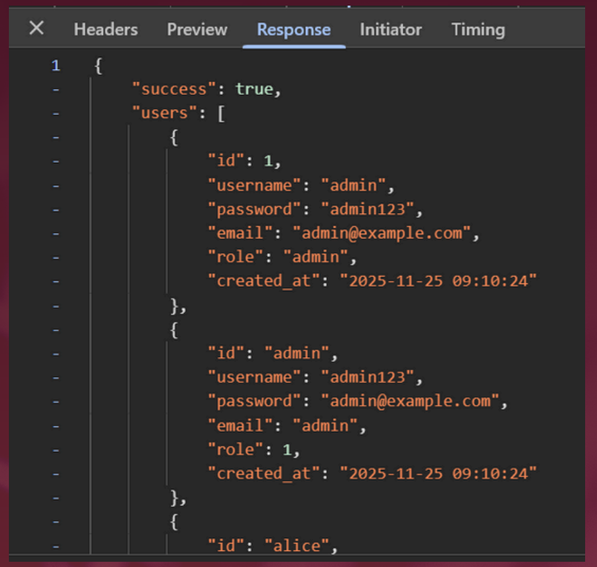
Terekspos semua username dan password dari database

User Enumeration

- User ID: 1 OR 1=1



- **User ID:** 1 UNION SELECT username, password, email, role, id, created_at FROM users



Dapat informasi semua user melalui ID lookup

SQL Injection **Remediation**

01 Parameterized Queries

```
const stmt = db.prepare('SELECT * FROM users WHERE
username = ? AND password = ?');
stmt.bind([username, password]);
const result = stmt.step() ? stmt.getAsObject() : null;
stmt.free();
```

Memisahkan logic SQL dari data input, memastikan input dari pengguna selalu diperlakukan sebagai data literal, bukan sebagai perintah eksekusi.

02 Input Sanitization

```
// Validasi username
function validateUsername(username) {
  // Hanya alphanumeric dan underscore
  const regex = /^[a-zA-Z0-9_]{3,20}$/;
  return regex.test(username);
}
```

Menerapkan pengecekan tipe data dan format untuk menolak input yang tidak valid.

03 Output Encoding

```
// Jangan expose password dalam response
const user = {
    id: row.id,
    username: row.username,
    email: row.email,
    role: row.role
  // Password tidak disertakan
};
```

Menyembunyikan data sensitif dari respons API.

04 Security Headers

```
import helmet from 'helmet';

app.use(helmet());

app.use((req, res, next) => {
    res.setHeader('X-Content-Type-Options', 'nosniff');
    res.setHeader('X-Frame-Options', 'DENY');
    res.setHeader('X-XSS-Protection', '1; mode=block');
    next();
});
```

Menggunakan middleware Helmet untuk menambah security headers.

SQL Injection
Remediation

05 Rate Limiting

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

const loginLimiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 5, // 5 attempts
    message: 'Too many login attempts, please try again later'
});
```

Membatasi jumlah attempt untuk melakukan login.

06 Error Handling yang Lebih Aman

```
try {
  // Query execution
} catch (error) {
    // Jangan expose error details
    console.error('Database error:', error);
    res.status(500).json({
    success: false,
    message: 'An error occurred. Please try again.'
    });
}
```

Tidak menggunakan payload dalam pesan error.

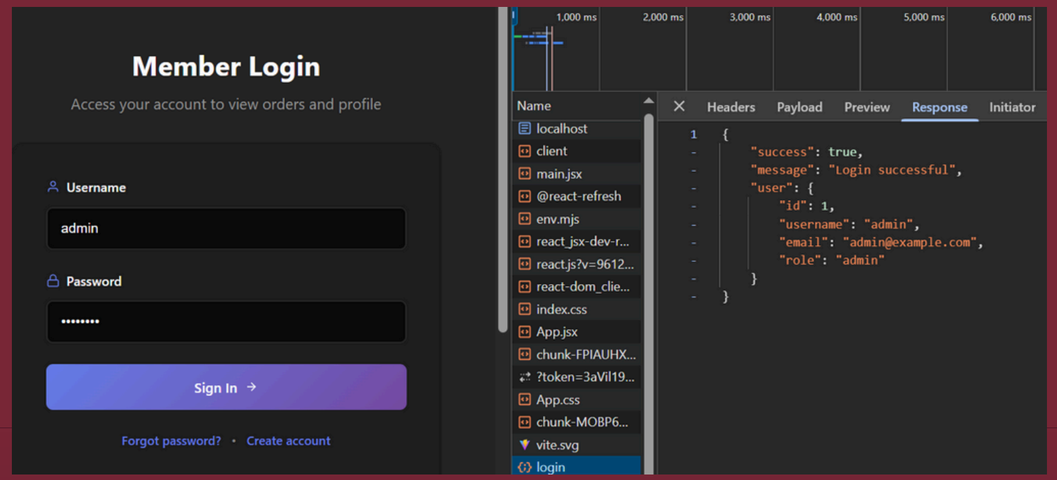
07 Password Hashing

```
import bcrypt from 'bcrypt';

// Saat registrasi
const hashedPassword = await bcrypt.hash(password, 10);

// Saat login
const match = await bcrypt.compare(password, user.password);
if (!match) {
    return res.status(401).json({ success: false, message: 'Invalid credentials' });
}
```

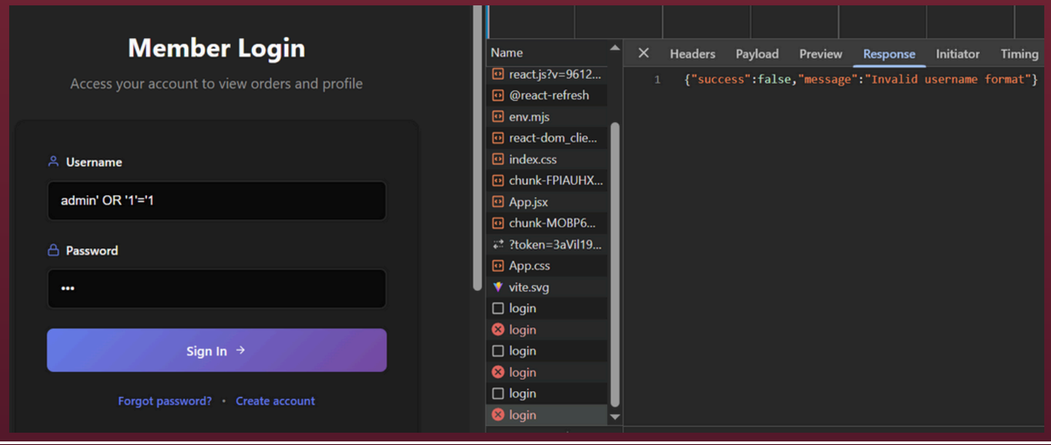
Mengecualikan data sensitif dari respons API.

Proof **SQL Injection**

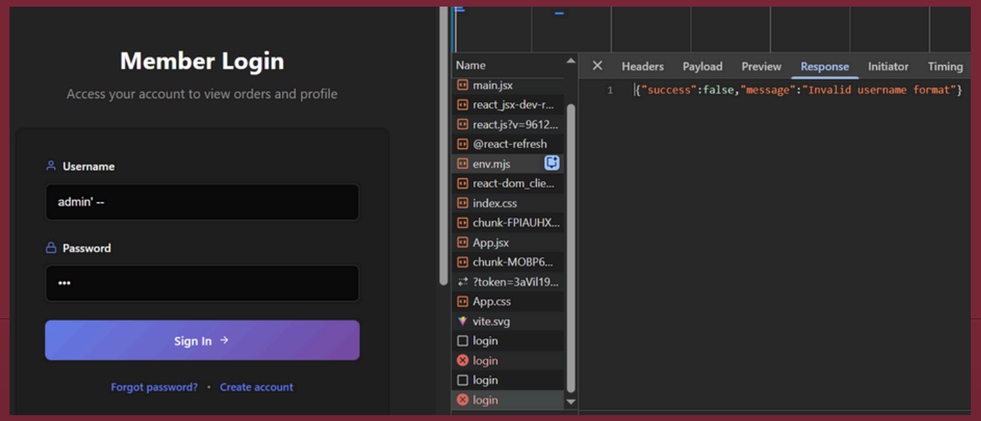
Login Biasa



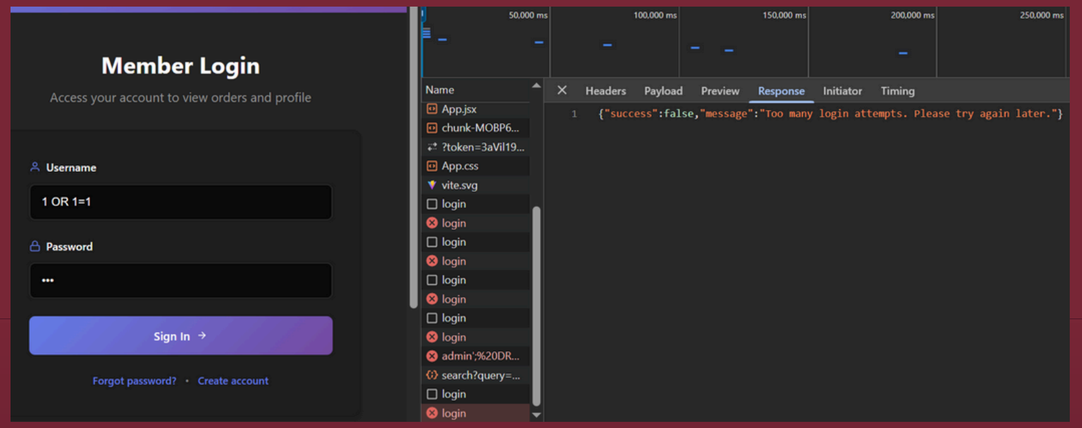
admin' OR '1'='1



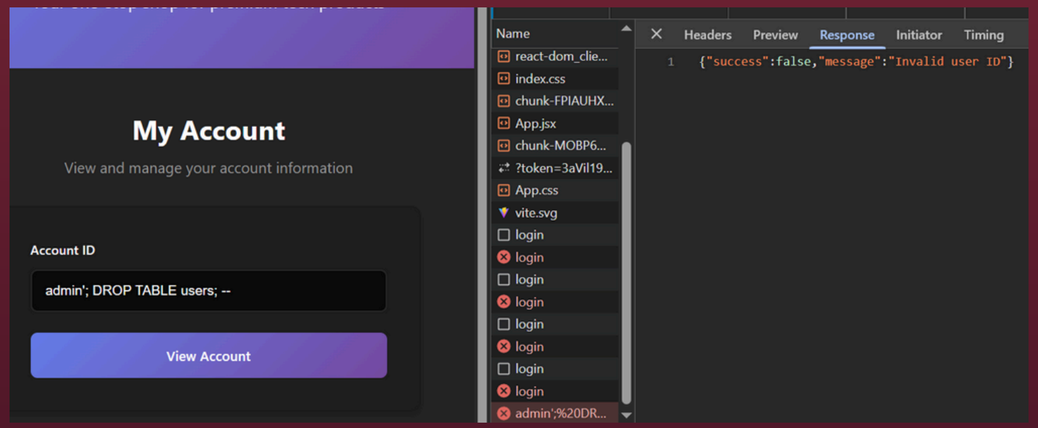
admin' --



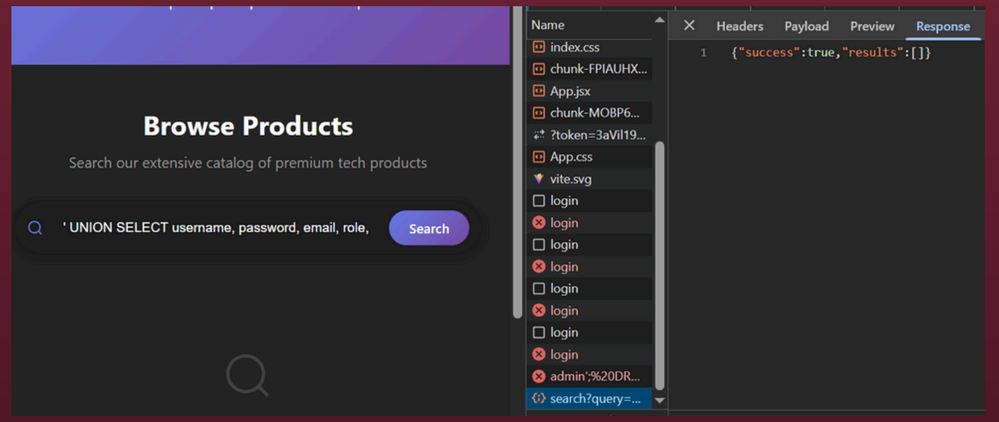
1 OR 1=1



admin'; DROP TABLE users; --



' UNION SELECT username, password, email, role, id, created_at FROM users --



THANK YOU

Github https://github.com/Tinkermannn/SQL-Command-Injection

Link Video https://youtu.be/B_yQ0tGB-zA