LAPORAN AKHIR PERANCANGAN KEAMANAN SISTEM DAN JARINGAN

ANALISIS DAN MITIGASI SERANGAN SQL INJECTION PADA APLIKASI WEB AKADEMIK MENGGUNAKAN CODEIGNITER 4

Dosen Pengampu: Ferdi Cahyadi S.Kom, M.Cs.

Anggota Kelompok :

Fito Anadiansyah – 2101020013

Rizqi Amanullah – 2301020002

Muhammad Arroyyan Hamel – 2301020117

Yudha Rifal Kelana – 2301020122

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

UNIVERSITAS MARITIM RAJA ALI HAJI

2024/2025

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Keamanan aplikasi web menjadi aspek krusial di era digital saat ini. Salah satu ancaman terbesar dan tertua yang masih sering terjadi adalah *SQL Injection* (SQLi). Serangan ini memungkinkan penyerang untuk memanipulasi kueri database, mengakibatkan kebocoran data sensitif, bypass autentikasi, hingga pengambilalihan sistem.

Proyek ini, yang diberi nama 'Glitch Academy', dirancang sebagai laboratorium simulasi untuk mendemonstrasikan bagaimana celah keamanan SQL Injection terjadi karena praktik pengkodean yang buruk (Vulnerable Mode) dan bagaimana cara menutup celah tersebut menggunakan fitur keamanan modern dari framework CodeIgniter 4 (Secure Mode).

1.2 Rumusan Masalah

1) Bagaimana mekanisme serangan SQL Injection dapat memanipulasi logika autentikasi untuk membobol halaman login tanpa password yang sah?
2) Bagaimana teknik Union-Based SQL Injection dapat dieksploitasi melalui fitur pencarian data untuk melakukan pencurian data sensitif (data exfiltration) dari tabel lain?
3) Bagaimana penerapan fitur Query Builder dan Model pada framework CodeIgniter 4 dapat memitigasi dan menutup celah keamanan SQL Injection dibandingkan dengan penggunaan Raw SQL?

1.3 Tujuan

1) Mendemonstrasikan serangan SQL Injection pada fitur Login (Authentication Bypass).
2) Mendemonstrasikan serangan Union-Based SQL Injection pada fitur Pencarian Data untuk pencurian data (Data Exfiltration).
3) Menerapkan teknik mitigasi menggunakan Query Builder pada CodeIgniter

1.4 Ruang Lingkup

Agar pembahasan dan implementasi proyek lebih terarah, penulis menetapkan batasan masalah sebagai berikut:

1) Objek Penelitian: Aplikasi web simulasi akademik bernama "Glitch Academy" yang dibangun menggunakan Framework CodeIgniter 4.
2) Jenis Serangan: Fokus analisis keamanan dibatasi hanya pada celah SQL Injection (SQLi), khususnya tipe Authentication Bypass dan Union-Based SQLi. Jenis serangan lain seperti XSS atau CSRF tidak dibahas.
3) Skenario Pengujian: Pengujian dilakukan dalam lingkungan lokal (localhost) menggunakan arsitektur "Dual-Mode", yaitu membandingkan kode yang sengaja dibuat rentan (Vulnerable Mode menggunakan Raw SQL) dengan kode yang aman (Secure Mode menggunakan Query Builder).
4) Teknik Mitigasi: Solusi keamanan yang diterapkan berfokus pada penggunaan fitur bawaan CodeIgniter 4, yaitu Query Builder dan Prepared Statements, tanpa menggunakan Web Application Firewall (WAF) pihak ketiga.

BAB II
DASAR TEORI

2.1 Deskripsi umum sistem

SQL Injection adalah teknik injeksi kode di mana penyerang menyisipkan perintah SQL berbahaya ke dalam input formulir web. Hal ini terjadi ketika aplikasi menggabungkan input pengguna secara langsung ke dalam string kueri SQL tanpa validasi atau sanitasi yang memadai.

2.2 Konsep Keamanan Informasi (CIA Triad)

Keamanan sistem informasi didasarkan pada tiga pilar utama yang dikenal sebagai CIA Triad (Confidentiality, Integrity, Availability). Menurut Albalawi et al. (2025), kerangka kerja ini sangat penting untuk mengidentifikasi risiko dan merancang protokol keamanan yang tepat [1]. Serangan SQL Injection dapat mengancam ketiga aspek ini:

1. Kerahasiaan (Confidentiality): SQL Injection memungkinkan penyerang mengakses data sensitif yang seharusnya tersembunyi. Hal ini sejalan dengan laporan OWASP yang menempatkan Injection sebagai salah satu ancaman keamanan web teratas [2].
2. Integritas (Integrity): Penyerang dapat memodifikasi atau menghapus data dalam basis data, merusak kepercayaan terhadap sistem [3].
3. Ketersediaan (Availability): Serangan intensif dapat menyebabkan Denial of Service (DoS) pada database.

2.3 Tipe serangan yang didemonstrasikan

A. Authentication Bypass: Memanipulasi logika WHERE pada kueri login sehingga selalu bernilai TRUE, memungkinkan akses tanpa password.
B. Union-Based SQLi: Menggunakan operator UNION untuk menggabungkan hasil dari dua kueri SELECT berbeda, memungkinkan penyerang melihat data dari tabel lain (misalnya tabel users) yang seharusnya tidak boleh diakses.

2.4 CodeIgniter 4 & Keamanan

CodeIgniter 4 menyediakan fitur keamanan bawaan seperti Escaping dan *Query Builder* yang secara otomatis menggunakan *Prepared Statements*, memisahkan data dari perintah SQL sehingga mencegah injeksi.

BAB III
PERANCANGAN SISTEM

Sistem dibangun dengan arsitektur ‘Dual-Mode’ dalam satu aplikasi:

Versi A (Vulnerable/Honeypot):

- Warna Tema: Hijau Neon.
- Metode: Menggunakan *Raw SQL* dengan penyambungan string manual (Concatenation).
- Tujuan**:** Sengaja dibuat rentan untuk diserang.

Versi B (Secure/Fortress):

- Warna Tema: Biru Neon.
- Metode: Menggunakan *CI4 Query Builder* dan *Model*.
- Tujuan: Menunjukkan kode yang aman dan kebal serangan.

3.1 Perbandingan Kode Fitur Login

Versi A:

Input langsung ditempel ke Query

$sql = "SELECT * FROM users WHERE nama_user = '$username'";

$query = $this->db->query($sql);

Analisis: Jika input berisi ‘ OR ‘1’=‘1, logika SQL menjadi benar untuk semua baris.

Versi B:

Menggunakan Model dan Binding

$user = $model->where('nama_user', $username)->first();

Analisis: Input dianggap sebagai string literal, bukan perintah SQL.

Perbandingan Kode Fitur Pencarian

Versi A:

$sql = ‘SELECT * FROM mahasiswa WHERE nama LIKE ‘%$keyword%’’;

Analisis: Memungkinkan penggunaan UNION SELECT untuk mengambil data dari tabel users.

Versi B:

$builder->like(‘nama’, $keyword);

Analisis: Karakter berbahaya otomatis di-escape oleh framework.

BAB IV
IMPLEMENTASI

Implementasi dari website yang belum dan sudah dimitigasi

```php
public function loginRawan(): RedirectResponse|string
{
    if (session()->get(key: 'isLoggedIn')) {
        return redirect()->to(uri: $this->getDashboardByRole(role: session()->get(key: 'role')));
    }
    return view(name: 'auth/login_rawan');
}

0 references | 0 overrides
public function authRawan(): RedirectResponse
{
    $session = session();

    $username = $this->request->getVar(index: 'nama_user');

    $sql = "SELECT * FROM users WHERE nama_user = '$username'";

    $query = $this->db->query(sql: $sql);
    $user  = $query->getRowArray();

    if ($user) {
        $session->setFlashdata(data: 'success', value: '⚠ LOGIN BERHASIL!');

        $this->setUserSession(user: $user);
        return redirect()->to(uri: $this->getDashboardByRole(role: $user['role']));
    } else {
        $session->setFlashdata(data: 'msg', value: 'Query gagal atau user tidak ditemukan. Coba payload lain.');
        return redirect()->to(uri: '/auth/login_rawan');
    }
}
```

Gambar 1 Sebelum Mitigasi

$sql = "SELECT * FROM users WHERE nama_user = '$username'";

$query = $this->db->query($sql);

Baris kode ini menunjukkan kondisi tidak aman karena Variabel $username masuk langsung ke string SQL.

```php
public function loginAman(): RedirectResponse|string
{
    if (session()->get(key: 'isLoggedIn')) {
        return redirect()->to(uri: $this->getDashboardByRole(role: session()->get(key: 'role')));
    }
    return view(name: 'auth/login_aman');
}

0 references | 0 overrides
public function authAman(): RedirectResponse
{
    $session = session();
    $model = new UserModel();

    $username = $this->request->getVar(index: 'nama_user');
    $password = $this->request->getVar(index: 'password');

    $user = $model->where(key: 'nama_user', value: $username)->first();

    if ($user) {
        $pass_verif = password_verify(password: $password, hash: $user['password']);

        if ($pass_verif) {
            $this->setUserSession(user: $user);
            return redirect()->to(uri: $this->getDashboardByRole(role: $user['role']));
        } else {
            $session->setFlashdata(data: 'msg', value: 'Password Salah!');
            return redirect()->to(uri: '/auth/login');
        }
    } else {
        $session->setFlashdata(data: 'msg', value: 'Username tidak ditemukan!');
        return redirect()->to(uri: '/auth/login');
    }
}
```

Gambar 2 Sesudah Mitigasi

$user = $model->where('nama_user', $username)->first();

Fungsi where() melakukan escaping otomatis.

if ($user && password_verify($password, $user['password']))

Fungsi hash untuk menambah keamanan

BAB V
PENGUJIAN & ANALISIS

5.1 Skenario 1: Login Bypass (Authentication)

- Target: Halaman Login Vulnerable (Hijau).
- Payload: ‘ OR ‘1’=‘1 (dimasukkan di kolom Username).
- Hasil: Sistem berhasil dibobol. Penyerang masuk sebagai user pertama di database (Admin) tanpa mengetahui password.
- Bukti: Dashboard terbuka dengan status ‘USER: admin’.

5.2 Skenario 2: Targeted Bypass (The Comment Assassin)

- Target: Halaman Login Vulnerable (Kolom Username).
- Tujuan: Login sebagai user ‘admin’ dengan mengabaikan bagian pengecekan password pada kueri SQL.
- Payload: admin’ –
- Hasil: Database hanya mengeksekusi pencarian user bernama ‘admin’ dan langsung memberikan akses login tanpa memverifikasi password.

5.3 Skenario 3: Union Based login (The Ghost User)

Ini adalah teknik tingkat lanjut di mana penyerang tidak login sebagai user yang sudah ada, melainkan memaksa database menciptakan ‘User Hantu’ yang datanya dipalsukan secara manual melalui kueri UNION.

- Syarat: Penyerang harus mengetahui jumlah kolom yang diminta oleh kueri login aplikasi (Diketahui ada 5 kolom: id, username, password, role, email).
- Payload: ‘ UNION SELECT 1, ‘sayahantu’, ‘123’, ‘admin’ - -

- Hasil di Mata Aplikasi: Kueri pertama (mencari user asli) akan gagal, lalu UNION menggabungkannya dengan data palsu. Aplikasi menerima data user dengan nama sayahantu dan role admin.
- Efek Demo: Di Dashboard akan tertulis: USER: sayahantu | ROLE: admin. Ini membuktikan aplikasi menerima sesi dari pengguna yang sebenarnya tidak ada di database asli.

5.4 Variasi Tautologi (The Boolean Brute)

Jika payload standar seperti ‘ OR ‘1’=‘1 diblokir oleh filter sederhana, penyerang dapat menggunakan variasi logika matematika atau string lain untuk mencapai tujuan yang sama: membuat pernyataan yang selalu bernilai BENAR (TRUE).

Payload: ‘ OR 100=100 - - atau ‘ OR ‘a’=’a

5.5 Pengujian pada secure mode

- Tindakan: Memasukkan payload serangan yang sama (Login Bypass dan Union Attack) pada halaman Login dan Dashboard versi Aman (Biru).
- Hasil:
- Login: Muncul pesan ‘Invalid Username or Password’.
- Search: Muncul pesan ‘NO DATA FOUND’. Database mencari mahasiswa yang namanya benar-benar mengandung karakter aneh tersebut, bukan mengeksekusinya sebagai perintah.

BAB VI
KESIMPULAN

Berdasarkan implementasi dan pengujian yang dilakukan, dapat disimpulkan bahwa:

1. Penggunaan *Raw SQL* dengan teknik penggabungan string (*string concatenation*) pada PHP sangat berbahaya dan membuka celah *SQL Injection* fatal.
2. Serangan *SQL Injection* tidak hanya bisa membobol login, tetapi juga bisa digunakan untuk mencuri seluruh isi database melalui teknik *UNION*.
3. Framework CodeIgniter 4 menyediakan mekanisme mitigasi yang efektif melalui *Query Builder* dan *Models*. Fitur ini memastikan input pengguna selalu disanitasi sebelum diproses oleh database, menutup celah keamanan sepenuhnya.

BAB VII
SARAN PENGEMBANGAN

7.1 Saran Pengembangan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, terdapat beberapa saran untuk pengembangan sistem keamanan "Glitch Academy" di masa mendatang:

1. Penerapan Web Application Firewall (WAF): Meskipun kode program sudah dimitigasi, disarankan untuk menambahkan WAF (seperti ModSecurity atau Cloudflare) sebagai lapisan pertahanan terluar untuk memblokir pola serangan injeksi sebelum mencapai aplikasi.
2. Mekanisme Logging dan Monitoring: Sistem perlu dilengkapi dengan fitur pencatatan log aktivitas (activity logging) yang dapat mendeteksi dan memberi peringatan kepada admin jika terjadi percobaan login yang mencurigakan secara berulang (brute force atau injection attempts).
3. Penggunaan Enkripsi SSL/TLS: Untuk memastikan keamanan data saat transmisi, aplikasi wajib diimplementasikan menggunakan protokol HTTPS agar username dan password tidak dapat disadap (sniffing) oleh pihak ketiga dalam jaringan.

BAB VIII
DAFTAR PUSTAKA

[1] N. Albalawi, N. Alamrani, R. Aloufi, dan A. R. Aljaedi, "A CIA Triad-Based Taxonomy of Prompt Attacks on Large Language Models," Systems, vol. 17, no. 3, p. 113, 2025.

[2] OWASP, "OWASP Top 10: 2021 The Ten Most Critical Web Application Security Risks," OWASP Foundation, 2021. [Online]. https://owasp.org/Top10/.

[3] M. D. F. Pratama, Songida, dan I. Gunawan, "Analisis Serangan dan Keamanan pada SQL Injection: Sebuah Review Sistematik," JIIFKOM (Jurnal Ilmiah Informatika dan Komputer), vol. 1, no. 2, hal. 27–32, 2022.