# Sistem Keamanan *Website* Dengan Multi Metode Untuk Mencegah *SQL Injection*

**Naomi Augusta[1], Asep Id Hadiana[2], Fajri Rakhmat Umbara[3]**
Program Studi Informatika
Universitas Jenderal Achmad Yani
JL Terusan Jend. Sudirman, Cibeber, Cimahi, Jawa Barat 405314, Indonesia
e-mail: [1]naomiaugusta08@gmail.com, [2]asep.hadiana@lecture.unjani.ac.id,
[3]fajri.rakhmat@lecture.unjani.ac.id

Correspondence : *e-mail*: naomiaugusta08@gmail.com

Diajukan: 15 Agustus 2024; Direvisi: 23 Agustus 2024; Diterima: 24 Agustus 2024

***Abstrak***

*Teknik serangan SQL Injection merupakan ancaman serius bagi keamanan aplikasi web, dengan 32% aplikasi web rentan terhadap serangan ini. Penelitian ini dilatarbelakangi oleh kekurangan pendekatan saat ini dalam menangani SQL Injection, khususnya terkait penggunaan SQL dinamis yang belum tepat. Penelitian sebelumnya menunjukkan bahwa kebanyakan aplikasi web rentan karena penerapan SQL dinamis, kurangnya validasi input, dan penanganan kesalahan yang tidak konsisten. Penelitian ini bertujuan mengusulkan empat metode untuk mencegah serangan SQL Injection pada sistem basis data aplikasi web. Proses dimulai dengan mengidentifikasi website yang rentan terhadap serangan SQL Injection, lalu dilakukan pencarian query yang terindikasi rentan terkena serangan SQL Injection. Query yang terindikasi rentan ini diperbaiki dengan menggunakan metode Input Validasi, PDO (PHP Data Object) Parameterized Query, Escape Characters, dan Stored Procedure. Hasil penerapan metode ini menunjukkan penurunan tingkat eksploitasi sebesar 100% dan peningkatan performa aplikasi hingga 93.75% lebih cepat dibandingkan dengan penerapan SQL dinamis tanpa metode pencegahan. Dengan demikian, penelitian ini diharapkan dapat meningkatkan lapisan pertahanan yang kuat dan efektif terhadap ancaman SQL Injection pada aplikasi web yang rentan, khususnya dalam konteks keamanan website dan database.*

***Kata kunci:*** *Escape characters,* Keamanan aplikasi web , PDO *parameterized query, SQL Injection*.

***Abstract***

*SQL Injection attack techniques are a serious threat to web application security, with 32% of web applications vulnerable to these attacks. This research is motivated by the shortcomings of current approaches in dealing with SQL Injection, specifically related to the improper use of dynamic SQL. Previous research has shown that most web applications are vulnerable due to the implementation of dynamic SQL, lack of input validation, and inconsistent error handling. This research aims to propose four methods to prevent SQL Injection attacks on web application database systems. The process starts with identifying websites that are vulnerable to SQL Injection attacks, then searching for queries that are indicated to be vulnerable to SQL Injection attacks. These vulnerable queries are repaired using Input Validation, PDO (PHP Data Object) Parameterized Query, Escape Characters, and Stored Procedure methods. The results of applying these methods show a decrease in the exploitation rate by 100% and an increase in application performance up to 93.75% faster than the application of dynamic SQL without prevention methods. Thus, this research is expected to increase a strong and effective defense layer against SQL Injection threats in vulnerable web applications, especially in the context of website and database security.*

***Keywords****: Escape characters, Web application security, PDO parameterized query, SQL Injection.*

## 1. Pendahuluan

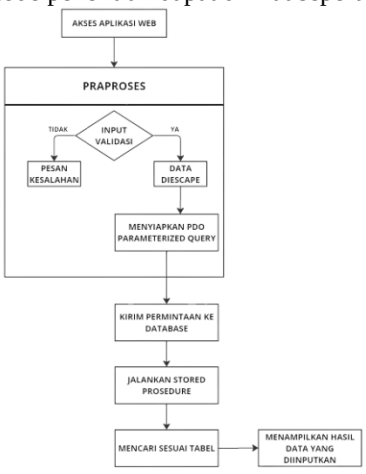
Keamanan aplikasi web telah menjadi perhatian utama seiring meningkatnya serangan siber, salah satunya adalah SQL *Injection*. Serangan ini memungkinkan penyerang menyisipkan perintah SQL berbahaya ke dalam kueri yang dieksekusi oleh basis data, sehingga mereka bisa mengakses data sensitif,

mengubah atau menghapus data, serta mengendalikan aplikasi sepenuhnya[1]. Kondisi ini tidak hanya mengancam integritas dan kerahasiaan data tetapi juga merusak reputasi organisasi yang menjadi korban.

Contoh nyata dari dampak besar serangan SQL *Injection* dapat dilihat pada kasus Marriott International. Pada tahun 2018, Marriott mengumumkan bahwa pihaknya telah mengalami pelanggaran data besar-besaran akibat serangan SQL *Injection* yang sebenarnya telah dimulai sejak tahun 2014. Serangan ini mengakibatkan kebocoran data pribadi sekitar 500 juta tamu yang menginap di hotel-hotel yang dikelola oleh Marriott[2]. Kasus ini menggambarkan betapa seriusnya ancaman SQL *Injection* terhadap keamanan data dan reputasi perusahaan, terutama dalam skala global. *Website* Penerimaan Peserta Didik Baru (PPDB) di SMAN 1 Parongpong ditemukan memiliki beberapa kelemahan keamanan yang membuatnya rentan terhadap SQL *Injection*. Masalah tersebut meliputi kurangnya validasi input pada form login, koneksi *database* yang tidak aman, serta penggunaan *query* yang tidak terparameterisasi pada berbagai halaman penting [3]. Hal ini memperbesar risiko serangan SQL *Injection* yang dapat mengancam keamanan data pengguna. Menurut laporan Veracode, 32% aplikasi web global rentan terhadap serangan SQL *Injection* [4], sementara di Indonesia, BSSN melaporkan bahwa 73% dari laporan kerentanan pada awal 2019 terkait dengan SQL *Injection* [5]. Dampak dari serangan ini bisa sangat merugikan, baik secara finansial maupun reputasi [6]. Penelitian sebelumnya mengidentifikasi bahwa penggunaan SQL dinamis tanpa validasi input yang memadai dan penanganan kesalahan yang buruk menjadi penyebab utama kerentanan ini [7]. Untuk mengatasi ancaman tersebut, metode pencegahan seperti PDO (PHP *Data Object*) *Parameterized Query*, *Stored Procedure*, validasi input, dan penggunaan escape characters telah diimplementasikan [8][9][10] Metode-metode ini bertujuan untuk mengurangi risiko serangan SQL *Injection* dengan memperkuat keamanan aplikasi. Penelitian ini berfokus pada sistem keamanan website SMAN 1 Parongpong dengan menggunakan penggabungan beberapa metode pencegahan untuk menciptakan lapisan keamanan yang lebih kuat. Pendekatan yang digunakan dalam penelitian ini melibatkan empat teknik utama yaitu,PDO *Parameterized Query*, validasi input, penggunaan *escape characters*, dan *Stored Procedure*. Dengan mengintegrasikan keempat teknik ini bertujuan untuk dapat mengatasi kerentanan *SQL Injection* secara lebih efektif dan meningkatkan keamanan aplikasi web.

## 2. Metode Penelitian

Penelitian ini bertujuan mengamankan website SMAN 1 Parongpong dari serangan SQL *Injection* dengan menggabungkan teknik Input Validasi, *PDO Parameterized Query*, *Escaping*, dan *Stored Procedure.* Website Penerimaan Peserta Didik Baru (PPDB) saat ini rentan karena kekurangan lapisan keamanan di berbagai halaman, seperti form login dan pengelolaan data. Langkah-langkah penelitian meliputi akses aplikasi oleh pengguna, validasi input, *escaping* data, penggunaan PDO *Parameterized Query*, pengiriman permintaan ke database, eksekusi *stored procedure*, pencarian data, dan akhirnya menampilkan hasil yang sesuai, metode penelitian dapat dilihat seperti pada gambar 1.




Gambar 1. Metode Penelitian.

### 2.1. Akses Aplikasi Web

Pada tahap awal, pengguna mengakses aplikasi web dengan memasukkan data melalui formulir di situs web, seperti formulir login atau pencarian. Pengguna memasukkan data seperti nama pengguna dan kata sandi di formulir login. Pengguna juga dapat memasukkan data lain yang dibutuhkan untuk mengakses atau mencari informasi di aplikasi web.

### 2.2. Praproses

Tahap praproses ini dilakukan validasi input, dimana tahap ini menjadi langkah awal penting dalam menghadapi berbagai tipe serangan SQL *Injection*, seperti *Union-based SQL Injection* dan *Boolean-based SQL Injection*. Validasi input adalah [9] penting dalam keamanan aplikasi yang bertujuan untuk memastikan bahwa data yang diterima dari pengguna memenuhi kriteria tertentu sebelum diproses lebih lanjut. Proses ini bertujuan untuk mencegah data yang tidak sah atau berbahaya dikirim ke sistem, yang bisa mengakibatkan serangan seperti SQL *Injection*. di mana aplikasi meemriksa semua data yang diterima dari pengguna resmi. Validasi input memastikan data hanya mengandung karakter yang sesuai dengan aturan yang ditetapkan seperti format data, tipe data, panjang data. Jika data tidak valid, aplikasi akan mengirimkan pesan kesalahan kepada pengguna untuk memperbaiki kesalahan input dan mencegah pemrosesan data yang tidak valid lebih lanjut dengan dilakukan escaping. Setelah validasi input menghasilkan data yang benar. Proses *escaping* dilakukan untuk mengubah karakter khusus yang berpotensi berbahaya menjadi entitas HTML yang aman dengan menggunakan fungsi seperti 'htmlspecialchars()'. Teknik ini sangat efektif dalam menghadapi serangan seperti *Tautologies SQL Injection* dan *Piggy-backed Queries*, di mana penyerang menggunakan karakter khusus seperti tanda kutip tunggal (') untuk memodifikasi kueri SQL. Proses ini mencegah serangan injeksi melalui karakter khusus dalam tahap input. Metode escape characters [11] adalah salah satu cara untuk melindungi aplikasi dari serangan ini dengan memastikan bahwa karakter-karakter khusus diinterpretasikan sebagai data bukan sebagai perintah SQL. Lalu menyiapkan *query* parameterisasi menggunakan PDO (PHP Data Objects). Teknik ini sangat efektif dalam menghadapi serangan *Blind SQL Injection* dan *Time-based SQL Injection*, di mana penyerang berusaha mendapatkan respons dari server tanpa langsung melihat hasilnya PDO memastikan *query* yang dikirim ke database aman dari injeksi SQL dengan menggantikan nilai langsung dalam *query* SQL dengan *placeholder* dan kemudian mengikat nilai actual ke placeholder tersebut. Placeholder adalah tanda dalam *query* SQL yang menunjukkan posisi nilai yang akan dimasukkan. Metode PDO *Parameterized Query* [6] menjadi langkah efektif untuk mencegah SQL *Injection* dengan menyederhanakan eksekusi *query* dan meningkatkan portabilitas kode pada berbagai basis data.

### *2.3. Stored Procedure*

*Stored procedure* berfungsi sebagai lapisan keamanan tambahan yang efektif dalam menghadapi serangan seperti *Error-based SQL Injection* dan *Out-of-band SQL Injection*. Metode ini, *server database* menjalankan *stored procedure* yang relevan setelah menerima permintaan dari aplikasi web. *Stored procedure* adalah [8] kumpulan perintah SQL yang disimpan di *server database*, yang berfungsi mengisolasi logika bisnis dan meningkatkan keamanan serta kinerja. Misalnya, dalam kasus *Error-based* SQL *Injection*, di mana penyerang memanfaatkan pesan kesalahan untuk mendapatkan informasi tentang struktur *database*, *stored procedure* memastikan bahwa semua logika terkait pengolahan data tetap berada di server, sehingga meminimalkan informasi yang bisa diekspos.

## 3. Hasil dan Pembahasan

Penelitian ini bertujuan untuk meningkatkan keamanan dan efisiensi aplikasi web terhadap serangan SQL *Injection* dengan menerapkan metode input validasi, PDO *Parameterized Query, Stored Procedure dan Escape Charactres*. Bagian ini akan memaparkan perbandingan *source code* sebelum dan sesudah penerapan metode, serta hasil pengujian performa yang dilakukan untuk mengevaluasi dampak dari metode tersebut.

### 3.1. *Sourcecode* Sebelum Dan Sesudah Menggunakan Metode

*Sourcecode* 1 yaitu, koneksi ke database: source code ini rentan terhadap SQL *Injection* karena penggunaan SQL dinamis tanpa sanitasi atau parameterisasi input. Selain itu, kurangnya metode keamanan pada koneksi dan penanganan kesalahan yang tidak memadai meningkatkan risiko serangan. *Sourcecode* 2 yaitu halaman kelola data peserta: Input dari variabel $_GET['id_peserta'] langsung digunakan dalam *query* SQL tanpa validasi, membuat aplikasi rentan terhadap SQL *Injection*. Penanganan kesalahan menggunakan die() juga tidak aman dan tidak memberikan informasi yang memadai. *Sourcecode* 3 yaitu halaman berkas siswa: penggunaan variabel $id_berkas tanpa sanitasi dalam *query* SQL membuka peluang untuk SQL *Injection*. Selain itu, terdapat redundansi dalam eksekusi *query* dan penanganan kesalahan yang tidak aman. *Sourcecode* 4 yaitu halaman jadwal tes: variabel $username digunakan langsung dalam *query* SQL tanpa sanitasi, meningkatkan risiko SQL *Injection*. Penanganan kesalahan menggunakan die() tidak aman, dan tidak ada validasi input yang layak. *Sourcecode* 5 yaitu halaman jadwal tes oleh peserta: Variabel $id_test digunakan dalam *query* SQL tanpa sanitasi, membuatnya rentan terhadap SQL *Injection*. Penanganan kesalahan tidak aman, dan kurangnya validasi input menambah risiko serangan. Sebagai contoh pada tabel 1 berisikan *sourcecode* sebelum dan sesudah menggunakan metode yang digunakan pada halaman kelola data peserta seperti pada tabel 1 dibawah.

Tabel 1. *Sourcecode* Sebelum dan Sesudah Menggunakan Metode

| Jenis Sourcecode | Sourcecode Sebelum | Sourcecode Sesudah |
|---|---|---|
| Sourcecode 2 | include "koneksi.php";<br>// Proses Pencarian<br>$id_peserta = isset($_GET['id_peserta']) ? $_GET['id_peserta'] : '';<br>$*query* = "SELECT id_peserta, nama_lengkap, username, email, no_hp, jenis_kelamin, tanggal_lahir, alamat, nilai_rata_rata, asal_sekolah FROM peserta";<br>if (!empty($id_peserta)) {<br>$*query* .= " WHERE id_peserta = '$id_peserta'";<br>}<br>$result = mysqli_*query*($koneksi, $*query*);<br>if (!$result) {<br>die("*Query* Error: " . mysqli_error($koneksi));<br>}<br>// Eksekusi *query*<br>$result = mysqli_*query*($koneksi, $*query*); | include "koneksinew.php";<br>// Proses Pencarian<br>$id_peserta = isset($_GET['id_peserta']) ? $_GET['id_peserta'] : '';<br>$*query* = "SELECT id_peserta, nama_lengkap, username, email, no_hp, jenis_kelamin, tanggal_lahir, alamat, nilai_rata_rata, asal_sekolah FROM peserta";<br>if (!empty($id_peserta)) {<br>$*query* = "CALL GetPeserta(:id_peserta)";<br>}<br>try {<br>$statement = $pdo->prepare($*query*);<br>if (!empty($id_peserta)) {<br>$statement->bindParam(':id_peserta', $id_peserta, PDO::PARAM_STR);<br>}<br>$statement->execute();<br>$data_peserta = $statement->fetchAll(PDO::FETCH_ASSOC);<br>} catch (PDOException $e) {<br>echo "*Query* Error: " . $e->getMessage();<br>die();<br>} catch (Exception $ex) {<br>echo "Error: " . $ex->getMessage();<br>die();<br>} |

### 3.2. Hasil Pengujian

Setelah menerapkan metode tersebut, dilakukan pengujian performa menggunakan SQL *Query Stress dan Microsoft SQL Server Client Statistics* untuk mengukur dampak penerapan metode terhadap waktu pemrosesan *query*.

#### 3.2.1. Pengujian Kemampuan Metode

Pengujian ini bertujuan untuk menilai efektivitas metode yang diusulkan dalam mencegah serangan *SQL Injection*. Pengujian dilakukan dengan menggunakan serangan yang disimulasikan terhadap website, baik secara manual maupun menggunakan alat seperti SQL Map. Hasil dari pengujian ini menunjukkan bahwa metode yang diusulkan berhasil mencegah eksploitasi SQL *Injection* yang tidak dapat dicegah oleh *query* SQL dinamis. Pengujian dilakukan untuk menguji efektivitas metode yang diusulkan dalam melindungi aplikasi web dari serangan SQL *Injection*. Simulasi serangan dilakukan dengan dua pengujian:

a. Test 1: Serangan pada web dengan *query* SQL dinamis.
b. Test 2: Serangan pada web setelah penerapan metode pengamanan.

Tabel 2. Hasil pengujian Kemampuan Metode

| SERANGAN | ... | ... | HASIL EKSPERIMEN 1 | HASIL EKSPERIMEN 2 |
|---|---|---|---|---|
| TAUTOLOGY | ... | ... | TEREKSPLOITASI | TIDAK TEREKSPLOITASI |
| LOGICALLY IMPRORER *QUERY* | ... | ... | TEREKSPLOITASI | TIDAK TEREKSPLOITASI |
| UNION *QUERY* | ... | ... | TEREKSPLOITASI | TIDAK TEREKSPLOITASI |
| PIGGY-BACKED QUERIES | ... | ... | TEREKSPLOITASI | TIDAK TEREKSPLOITASI |
| SQL MAP | ... | ... | TEREKSPLOITASI | TIDAK TEREKSPLOITASI |

### 3.2.2. Pengujian Waktu Pemrosesan

Pengujian ini bertujuan untuk mengukur efektivitas metode dalam hal waktu pemrosesan *query*. Pengujian dilakukan menggunakan dua alat: *Microsoft SQL Server Client Statistics* dan SQL *Query* Stress. Hasilnya menunjukkan bahwa metode yang diusulkan menghasilkan waktu pemrosesan yang lebih efisien dibandingkan dengan metode yang menggunakan SQL dinamis, terutama dalam hal pemakaian CPU dan waktu eksekusi *query*.

### *3.2.3. Microsoft SQL Server Client Statistics*

Pada penelitian ini penggunaan instrument tersebut untuk mengevaluasi kinerja kueri yang dieksekusi melalui aplikasi web yang rentan terhadap serangan SQL *Injection*. Dengan membandingkan statistic kinerja antara kueri yang aman dan kueri yang dieksploitasi oleh SQL *Injection*. Bertujuan untuk menilai apakah SQL berperan dalam meningkatkan beban kerja lalu lintas. Selama evaluasi, waktu eksekusi dan pemrosesan klien akan didokumentasikan seperti pada tabel 3 dibawah ini.

Tabel 3. Pengujian Waktu Pemrosesan menggunakan *Microsoft SQL Server Client Statistic*

| KRITERIA EVALUASI | ... | ... | HALAMAN KELOLA DATA PESERTA | | HALAMAN JADWAL PESERTA | | HALAMAN LOGIN | | HALAMAN BERKAS SISWA | | HALAMAN JADWAL TEST | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Test 1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 |
| WAKTU PEMROSESAN KLIEN | ... | ... | 1.5 | 0.8 | 2.1 | 0.3 | 4.8 | 0.3 | 15.1 | 10.9 | 12.5 | 2.9 |
| WAKTU REPONS SERVER | ... | ... | 0.3 | 0.5 | 0.3 | 0.9 | 0.9 | 1.1 | 0.7 | 0.6 | 0.8 | 1.0 |
| TOTAL WAKTU EKSEKUSI | ... | ... | 1.8 | 1.3 | 2.4 | 1.2 | 5.7 | 1.4 | 15.8 | 11.5 | 13.3 | 3.9 |

Hasil pengujian *Microsoft SQL Server Client Statistic* menunjukkan bahwa penerapan *Stored Procedure* dan metode keamanan lainnya secara konsisten meningkatkan efisiensi pemrosesan dan mengurangi waktu eksekusi dibandingkan dengan *query* SQL biasa. Pengujian pada lima halaman aplikasi web menunjukkan penurunan signifikan dalam waktu pemrosesan klien dan total waktu eksekusi. Halaman kelola data peserta mengalami pengurangan total waktu eksekusi sebesar 27.8%. Halaman jadwal peserta menunjukkan penurunan total waktu eksekusi sebesar 50%, dengan waktu pemrosesan klien menurun sebesar 85.71%. Pada halaman login, waktu pemrosesan klien berkurang 93.75% dan total waktu eksekusi turun 75.44%. Halaman berkas siswa mencatat penurunan total waktu eksekusi sebesar 27.22%, dengan efisiensi pemrosesan klien meningkat 27.81%. Terakhir, halaman jadwal tes menunjukkan penurunan total waktu eksekusi yang signifikan sebesar 70.7% dan waktu pemrosesan klien berkurang 76.8%. Secara keseluruhan, hasil ini membuktikan bahwa penggunaan *stored procedure* dan teknik keamanan lainnya secara efektif meningkatkan kinerja aplikasi web, terutama dalam hal pengurangan waktu pemrosesan klien dan total waktu eksekusi.

### *3.2.4.* SQL *Query Statics*

Penelitian ini menggunakan perangkat lunak SQL*QueryStress* sebagai alat untuk mengevaluasi kinerja *query* saat diberikan beban kerja yang signifikan. Tujuan utamanya adalah untuk meneliti efek dari berbagai jenis *query* SQL terhadap kinerja sistem. Perbedaan struktural antara *query* sebelum dan sesudah menggunakan metode menyebabkan cara eksekusi yang berbeda. Hasil pengujian dapat dilihat seperti pada tabel 4 dibawah ini.

Tabel 4 Pengujian Waktu Pemrosesan menggunakan SQL *Query* Stress

| KRITERIA | ... | HALAMAN KELOLA DATA PESERTA | | HALAMAN JADWAL PESERTA | | HALAMAN LOGIN | | HALAMAN BERKAS SISWA | | HALAMAN JADWAL TEST | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Test1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 | Test 1 | Test 2 |
| Elapsed | ... | 01:41:3598 | 01:41:3983 | 01:41.0637 | 01:41.1571 | 01:41.3985 | 01:41.3825 | 01:41:4261 | 01:41:3595 | 01:41:4062 | 01:41:1732 |
| CPU Seconds | ... | 0,0004 | 0,0002 | 0,0002 | 0,0001 | 0,0001 | 0,0002 | 0,0001 | 0,0001 | 0,0001 | 0,0000 |
| Actual Seconds | ... | 0,0043 | 0,0033 | 0,0034 | 0,0032 | 0,0043 | 0,0034 | 0,0030 | 0,0060 | 0,0032 | 0,0042 |
| Client Seconds | … | 0,0030 | 0,0029 | 0,0014 | 0,0012 | 0,0032 | 0,0034 | 0,0043 | 0,0044 | 0,0045 | 0,0022 |
| Logical Reads | … | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 2,0000 |

Hasil pengujian SQL *Query Stress* pada lima halaman aplikasi menunjukkan bahwa penggunaan *stored procedure* umumnya lebih efisien dalam meningkatkan kinerja sistem. Pada halaman kelola data peserta dan jadwal peserta, *stored procedure* mempercepat penggunaan CPU dan waktu eksekusi aktual, meskipun *elapsed time, client seconds*, dan *logical reads* hampir sama dengan *query* biasa. Halaman *login* menunjukkan sedikit peningkatan waktu penggunaan CPU dengan *stored procedure*, namun eksekusi aktual lebih cepat. Halaman berkas siswa menunjukkan bahwa meskipun waktu eksekusi aktual per iterasi lebih tinggi, *elapsed time* lebih rendah dengan *stored procedure*, menandakan efisiensi keseluruhan yang lebih baik. Pada halaman jadwal tes, *stored procedure* memberikan kinerja terbaik dalam total waktu eksekusi, performa klien, dan penggunaan CPU, menjadikannya pilihan yang lebih efektif untuk meningkatkan efisiensi sistem.

## 4. Kesimpulan

Kesimpulan pada penelitian ini berhasil menguji pengamanan terhadap serangan *SQL Injection* pada sebuah website, dimana keberhasilan metode pengamanan yang diusulkan yaitu, input validasi, PDO *parameterized query, stored procedure dan escaping* terbukti efektif dalam melindungi aplikasi web dari serangan SQL *Injection*. Simulasi serangan dengan *query* SQL dinamis pada uji pertama menunjukkan kerentanan, sedangkan uji kedua dengan metode pengamanan menunjukkan tidak ada serangan yang berhasil. Pengujian efisiensi menggunakan *Microsoft Client Statistic dan SQL Query Stress* menunjukkan bahwa *stored procedure* lebih efisien, mengurangi waktu pemrosesan klien dan total waktu eksekusi, meskipun ada sedikit peningkatan pada *wait time on server replies*. Pengujian pada lima halaman aplikasi menunjukkan peningkatan performa signifikan dengan *stored procedure*. Penelitian lanjutan disarankan untuk menguji metode keamanan tambahan, mengoptimalkan indeks database, serta mempertimbangkan *caching*, *load balancing*, dan optimasi konfigurasi server dan jaringan.

## Daftar Pustaka

[1] Y. Yulianingsih, "Menangkal Serangan SQL Injection Dengan Parameterized Query," *J. Edukasi dan Penelit. Inform.*, vol. 2, no. 1, pp. 46–49, 2016, doi: 10.26418/jp.v2i1.15507.

[2] O. W. Purbo, "Mid 2020 Cyber Security Threat, Tips dan Proposal Strategi Mitigasi Nasional," *Lms.Onnocenter.or.Id*, 2021, [Online]. Available: https://lms.onnocenter.or.id/pustaka/docs/internet-indonesia/tiktok-id-wp-mid-2020-report.pdf

[3] M. L. Siddiq, M. R. R. Jahin, M. R. Ul Islam, R. Shahriyar, and A. Iqbal, "SQLIFIX: Learning Based Approach to Fix SQL Injection Vulnerabilities in Source Code," *Proc. - 2021 IEEE Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2021*, pp. 354–364, 2021, doi: 10.1109/SANER50967.2021.00040.

[4] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, "A CNN-bilstm based Approach for Detection of SQL Injection Attacks," *Proc. 2nd IEEE Int. Conf. Comput. Intell. Knowl. Econ. ICCIKE 2021*, pp. 378–383, 2021, doi: 10.1109/ICCIKE51210.2021.9410675.

[5] K. Ahmad and M. Karim, "A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 6, pp. 324–332, 2021, doi: 10.14569/IJACSA.2021.0120636.

[6] M. Sendiang, A. Polii, and J. Mappadang, "Minimization of SQL injection in scheduling application development," *2016 Int. Conf. Knowl. Creat. Intell. Comput. KCIC 2016*, pp. 14–20, 2016, doi: 10.1109/KCIC.2016.7883619.

[7] L. K. Shar and H. B. K. Tan, "Defeating SQL injection," *Computer (Long. Beach. Calif).*, vol. 46, no. 3, pp. 69–77, 2013, doi: 10.1109/MC.2012.283.

[8] Z. C. S. S. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks," *2020 IEEE Conf. Comput. Appl. ICCA 2020*, pp. 1–6, 2020, doi: 10.1109/ICCA49400.2020.9022833.

[9] V. Abdullayev and D. A. S. Chauhan, "SQL Injection Attack: Quick View," *Mesopotamian J. Cyber Secur.*, vol. 2023, pp. 30–34, 2023, doi: 10.58496/mjcs/2023/006.

[10] Aprilia Monica Sari, Trihana Santhi, Dewa Ketut Alit Maha Putra, Muhamad Bintang Haekal, I Made Edy Listartha, and Gede Arna Jude Saskara, "Pengukuran Efektivitas Serangan Sql Injection Pada Website Dengan Menggunakan Tools Jsql, Havij, Dan the Mole," *J. Inform. Dan Tekonologi Komput.*, vol. 3, no. 1, pp. 35–42, 2023, doi: 10.55606/jitek.v3i1.905.

[11] L. Ma, D. Zhao, Y. Gao, and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web," *Proc. - 2nd Int. Conf. Comput. Network, Electron. Autom. ICCNEA 2019*, pp. 176–179, 2019, doi: 10.1109/ICCNEA.2019.00042.