Terbit *online* pada laman : http://teknosi.fti.unand.ac.id/

Jurnal Nasional Teknologi dan Sistem Informasi

| ISSN (Print) 2460-3465 | ISSN (Online) 2476-8812 |

Artikel Penelitian

Penetration Testing pada Kerentanan Keamanan Sistem PELAKAT Menggunakan *SQL Injection*

Khairul [a], *Asrul Abdullah* [b,*], *Sucipto* [c]

[a,b,c] *Teknik Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Muhammadiyah Pontianak, Jl. Ahmad Yani No.111, Pontianak 78123, Indonesia*

INFORMASI ARTIKEL

Sejarah Artikel:
Diterima Redaksi: 15 Maret 2025
Revisi Akhir: 08 Mei 2025
Diterbitkan *Online*: 13 Mei 2025

KATA KUNCI

Penetration Testing,
SQL Injection,
Burp Suite,
SQLMap,
Kerentanan Sistem

KORESPONDENSI

E-mail: asrul.abdullah@unmuhpnk.ac.id*

A B S T R A C T

Penetration Testing bertujuan untuk mengidentifikasi kerentanan sistem dengan cara mensimulasikan serangan dengan teknik tertentu seperti *SQL Injection*. Sistem Pelayanan Administrasi Kependudukan yang Mendekatkan Masyarakat (PELAKAT) adalah sebuah aplikasi berbasis *website* yang dibuat oleh Dinas Kependudukan dan Pencatatan Sipil (Disdukcapil) Kabupaten Sambas untuk memudahkan proses pengelolaan beberapa dokumen administrasi kependudukan (Adminduk). Pengujian keamanan sistem PELAKAT menggunakan teknik *SQL Injection* diperlukan untuk mengidentifikasi kerentanannya serta memberikan rekomendasi mitigasi. Tahapan metode *penetration testing* yang dilakukan yaitu *reconnaissance*, *scanning*, *vulnerability assessment*, *exploitation*, dan *reporting*. *Tools* yang digunakan yaitu *Burp Suite* untuk menganalisis HTTP *request* dan *SQLMap* untuk eksploitasi kerentanan. Berdasarkan hasil pengujian, salah satu parameter pada *form* login sistem PELAKAT diketahui rentan terhadap *SQL Injection*. Eksploitasi berhasil mengakses sembilan *database*, lima tabel pada salah satu *database*, dan 13 kolom pada salah satu tabel. Kerentanan ini disebabkan karena sistem dikembangkan tanpa fitur keamanan yang memadai. Tingkat kerentanan sistem dinilai tinggi karena sistem PELAKAT dinyatakan rentan terhadap *SQL Injection* sehingga diperlukan tindakan mitigasi. Rekomendasi mitigasi meliputi penerapan *WAF* (*Web Application Firewall*), validasi input pada *form* input, penggunaan *prepared statements*, implementasi *framework* seperti Laravel, dan migrasi *database* ke penyimpanan berbasis *cloud*. Dengan penerapan mitigasi ini, diharapkan dapat meningkatkan keamanan sistem dan meminimalisir kerentanan sistem.

1. PENDAHULUAN

Pada era saat ini, sistem informasi sangat diperlukan di berbagai bidang termasuk bidang pemerintahan. Sistem Pelayanan Administrasi Kependudukan yang Mendekatkan Masyarakat (PELAKAT) merupakan salah satu contoh sistem informasi berbasis *website* yang diterapkan oleh Dinas Kependudukan dan Pencatatan Sipil (Disdukcapil) Kabupaten Sambas yang digunakan oleh petugas Disdukcapil Sambas, operator kecamatan, dan operator desa. Sistem ini bertujuan untuk meningkatkan pelayanan adminduk kepada masyarakat Kabupaten Sambas serta pengelolaan data adminduk secara digital.

Namun semakin berkembangnya teknologi saat ini juga memberikan dampak negatif seperti serangan eksploitasi dan penyalahgunaan data pada suatu sistem oleh pihak yang tidak bertanggung jawab. Serangan tersebut dapat terjadi pada sistem jika keamanan sistem memiliki celah [1]. Contoh akibat dari serangan tersebut yaitu berupa kehilangan data hingga kebocoran data [2]. Maka dari itu diperlukan pengujian perangkat lunak pada sistem PELAKAT untuk memastikan keamanan datanya sehingga dapat meminimalisir risiko tersebut.

Pengujian perangkat lunak merupakan proses membuktikan dan mengamati cara kerja sistem agar sesuai dengan fungsinya [3]. Pengujian perangkat lunak bertujuan untuk menjamin kualitas

https://doi.org/10.25077/TEKNOSI.v11i1.2025.78-86

sistem. Kualitas suatu sistem dapat dikatakan terjamin jika *bug* yang teridentifikasi dapat diminimalisir dan keamanan sistemnya tidak mudah ditembus oleh serangan yang dapat merugikan pihak terkait.

Berdasarkan informasi dari Disdukcapil Sambas, sistem PELAKAT pernah mengalami masalah terhadap keamanan sistemnya yang menyebabkan kehilangan data pada *database* sistem PELAKAT. Menurut laporan dari pihak Disdukcapil Sambas, masalah tersebut merupakan sebuah pelanggaran keamanan sistem yang berupa aktivitas ilegal dari pihak yang tidak bertanggung jawab. Sementara ini, Disdukcapil Sambas sudah melakukan beberapa upaya seperti memulihkan *database* menggunakan data *backup* serta membatasi akses sistem PELAKAT menggunakan *VPN* (*Virtual Private Network*). Berdasarkan keterangan tersebut, implementasi *penetration testing* menggunakan teknik *SQL Injection* pada sistem PELAKAT merupakan pengujian yang dapat dipertimbangkan untuk menguji kerentanan keamanannya.

Berdasarkan penelitian yang berjudul "Analisis Keamanan *Website* Leads UPNVJ Terhadap Serangan *SQL Injection* & *Sniffing Attack*". *SQL Injection* merupakan teknik pengujian keamanan *website* menggunakan perintah *SQL* untuk menembus *database* sehingga dapat melihat dan mengubah data sensitif pada sistem. Oleh karena itu, jika ada pihak yang tidak bertanggung jawab melakukan *SQL Injection* secara ilegal untuk mengakses *database website* maka dapat mengakibatkan ancaman keamanan serius pada *website* tersebut [4]. Bersumber dari *The Open Worldwide Application Security Project* (*OWASP*), 94% aplikasi diuji dalam beberapa bentuk *injection* seperti *SQL Injection* yang merupakan salah satu dari sepuluh kerentanan teratas pada *website* [5]. Maka dengan implementasi pengujian tersebut dapat mengidentifikasi celah keamanan sistem lebih awal sehingga dapat mengurangi risiko kehilangan hingga kebocoran data.

Aplikasi berbasis *website* adalah sebuah program perangkat lunak yang dikembangkan menggunakan bahasa pemrograman seperti HTML, PHP, CSS, dan JavaScript. Aplikasi berbasis *website* memerlukan *web server* dan *browser* seperti *Chrome* atau *Firefox* untuk mengoperasikannya serta memerlukan jaringan internet ataupun intranet [6]. Aplikasi berbasis *website* terdiri atas dua jenis yaitu statis dan dinamis. Aplikasi berbasis *website* statis terdiri dari halaman-halaman yang kontennya tidak dapat diubah oleh pengguna. Sedangkan aplikasi berbasis *website* dinamis mampu menampilkan halaman-halaman yang dapat berubah berdasarkan interaksi pengguna. *Website* dinamis dapat dikembangkan dengan hanya menggunakan bahasa pemrograman PHP tanpa *framework* (*PHP Native*) dan dapat dikembangkan menggunakan kerangka kerja (*framework*) seperti Laravel dan CodeIgniter. Berdasarkan dari segi keamanan, *website* yang dikembangkan menggunakan *PHP Native* lebih rentan dibandingkan *website* yang dikembangkan menggunakan Laravel. Hal tersebut karena *website* yang dikembangkan menggunakan *PHP Native* tidak menyediakan fitur keamanan bawaan sehingga pengembang diharuskan menerapkan fitur keamanan secara manual sehingga memerlukan pengembangan yang lebih mandiri dari awal. Berbeda dengan *website* yang dikembangkan menggunakan *framework* seperti Laravel yang telah menyediakan fitur keamanan bawaan sehingga pengembangan lebih terstruktur.

Sistem Pelayanan Administrasi Kependudukan yang Mendekatkan Masyarakat (PELAKAT) adalah sebuah aplikasi berbasis *website* yang dibuat oleh Dinas Kependudukan dan Pencatatan Sipil (Disdukcapil) Kabupaten Sambas. Sistem ini digunakan oleh petugas Disdukcapil Sambas, operator kecamatan, dan operator desa. Aplikasi ini bertujuan untuk memudahkan proses pengelolaan beberapa dokumen administrasi kependudukan (Adminduk) masyarakat di seluruh kecamatan dan desa di Kabupaten Sambas secara *online* dan digital yang terintegrasi ke Disdukcapil Sambas.

Software Quality Assurance atau penjaminan kualitas perangkat lunak adalah serangkaian kegiatan yang terstruktur yang dimaksudkan untuk memberikan keyakinan melalui penilaian dan evaluasi proses dalam pembuatan atau pengembangan perangkat lunak sesuai dengan mutu dan fungsionalnya yang telah ditetapkan [7]. *Software Quality Assurance* bertujuan untuk memastikan perangkat lunak memenuhi ekspektasi pengguna dan memenuhi standar kualitas sehingga dapat mengurangi biaya pengembangan dan mengurangi risiko perangkat lunak berkualitas rendah. Penjaminan perangkat lunak dapat dilakukan dengan cara melakukan *software testing*.

Software Testing atau pengujian perangkat lunak adalah suatu aktivitas yang dilakukan untuk memastikan bahwa fungsi perangkat lunak telah diterapkan dengan benar sesuai tujuannya, menemukan dan mengurangi *bug*, sehingga kekurangan tersebut dapat diperbaiki dan siap digunakan oleh pengguna [8]. Pengujian perangkat lunak dapat dilakukan dari aspek fungsional dan non-fungsional. Contoh pengujian berdasarkan aspek fungsional ini yaitu pengujian fungsi login pengguna dan navigasi. Sementara itu, contoh dari pengujian berdasarkan aspek non-fungsional yaitu pengujian kinerja dan keamanan sistem. Salah satu contoh pengujian keamanan sistem yaitu *penetration testing*.

Penetration testing atau pengujian penetrasi adalah proses evaluasi keamanan sistem perangkat lunak dengan cara mensimulasikan serangan dari pengguna yang tidak berwenang masuk secara ilegal ke dalam sistem menggunakan *tool* pendukung ataupun secara manual [9]. Salah satu jenis *penetration testing* yaitu *penetration testing* aplikasi berbasis *website*. Tujuan dari *penetration testing* aplikasi berbasis *website* yaitu mengidentifikasi kerentanan sistem sehingga keamanan sistem dapat diperbaiki lebih awal oleh pihak pengembang sebelum penyerang sebenarnya dapat menyalahgunakan celah keamanan sistemnya. *Penetration testing* mempunyai beberapa teknik seperti *Broken Access Control*, *Cryptographic Failures*, dan *SQL Injection*.

SQL Injection adalah teknik peretasan dengan cara mengubah perintah SQL yang dapat dilakukan pada aplikasi berbasis *website* yang penyimpanan datanya menggunakan *database* [10]. *SQL Injection* merupakan salah satu teknik *penetration testing* pada aplikasi berbasis *website*. Teknik *SQL Injection* memungkinkan untuk manipulasi *query* SQL *database* sistem. Gambar 1 merupakan salah satu contoh *query* SQL yang dapat dimanipulasi dan rentan.

```
$login = "SELECT * FROM user WHERE email = '$email'";
```

Gambar 1. Contoh *Query* SQL Rentan

Input pengguna pada *query* tersebut tidak hanya diproses sebagai *value* dari parameter, melainkan diproses dan dianggap sebagai bagian dari perintah SQL. Oleh karena itu, *query* SQL dapat menjadi rentan jika *value* diubah menjadi simbol tertentu seperti tanda petik tunggal yang dapat menyebabkan *query* SQL error. Penggunaan *prepared statements* pada *query* SQL dapat meminimalisir kerentanan. *Prepared statements* dapat menggunakan "*prepare*()" dan "*bind_param*()" pada *query* SQL. Penggunaan *prepared statements* akan memisahkan *value* yang diinput pengguna dari perintah SQL, sehingga input pengguna tidak dianggap sebagai bagian dari perintah SQL. Gambar 2 merupakan contoh implementasi *prepared statements*.

```
$prestat = prepare("SELECT * FROM user WHERE email = :email");
$prestat->bind_param(':email', $email);
```

Gambar 2. Implementasi *Prepared Statements*

SQL Injection dapat berdampak pada kebocoran data sensitif, eksploitasi data, dan bahkan pengambil alihan sistem *database*. Implementasi *SQL Injection* dapat dilakukan menggunakan *tools Burp Suite* dan *SQLMap*.

Burp Suite adalah sebuah *tool* untuk melakukan pengujian keamanan pada aplikasi *website* yang dikembangkan oleh *PortSwigger Security*. Tool ini banyak digunakan karena memungkinkan dapat melakukan pengotomatisan pemindaian kerentanan pada aplikasi *website* yang diuji dan memeriksa serangan yang mungkin dapat terjadi pada *website* tersebut. Salah satu fitur dari *Burp Suite* yaitu *proxy service* yang berfungsi untuk mencegat (*intercept*) permintaan HTTP dari *web browser* ke server [11]. Melalui *intercept* HTTP request sistem, *Burp Suite* dapat melakukan analisis respon server jika sistem memiliki kerentanan terhadap *SQL Injection*.

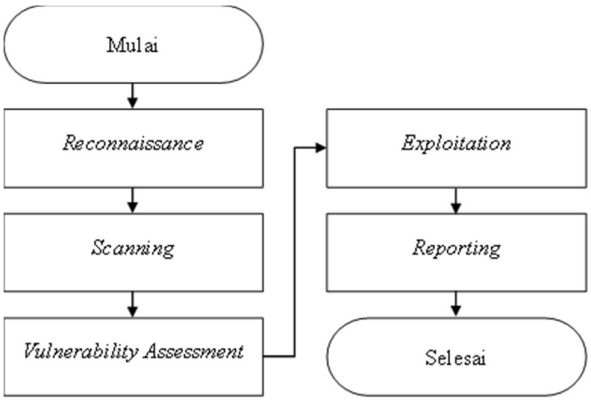
SQLMap adalah sebuah *tool penetration testing open source* yang mengotomatiskan proses pendeteksian dan eksploitasi kelemahan *SQL Injection* serta pengambilalihan server *database*. *Tool* ini dilengkapi dengan banyak fitur khusus untuk *penetration testing* seperti pengambilan data dari *database* dan mengakses *file system* [12]. *SQLMap* merupakan *tool* yang dibuat menggunakan bahasa pemrograman python dan *tool* ini dijalankan melalui terminal komputer atau *CMD* (*Command Prompt*). Beberapa fitur pada *SQLMap* yaitu dapat terhubung langsung ke *database* serta dapat mengunduh dan mengunggah *file* jika *database* sistem yang diuji menggunakan *MySQL*, *PostgreSQL*, atau *Microsoft SQL Server*.

Adapun beberapa penelitian terdahulu yang berkaitan dengan penelitian ini yaitu penelitian oleh [13] yang membahas *penetration testing* pada sepuluh *website* secara acak menggunakan beberapa teknik seperti *brute force* dan *bypass* serta *tools Hydra* dan *Metasploit*. Hasil penelitian ini menyatakan bahwa 80% dari *website* yang diuji memiliki kelemahan terhadap serangan *SQL injection*. Kemudian, penelitian [14] membahas perbandingan tiga *tools SQL Injection* yaitu *SQLMap*, *SQLsus*, dan *The Mole* dengan mempertimbangkan tiga parameter seperti *Cross Program*, *Functionality*, *Usability*. Berdasarkan hasil pengujiannya, diketahui bahwa *SQLMap* memiliki beberapa keunggulan sehingga pada proses *exploitation* metode *penetration testing* pada sistem PELAKAT menggunakan *tool SQLMap*. Selain itu, pada penelitian [15] membahas penganalisisan perbandingan *tools OWASP*, *Nikto*, dan *Burp Suite* berdasarkan aspek objek scan, durasi, kemudahan, dan informasi pada pemindaian kerentanan (*vulnerability scanning*). Hasil penelitiannya menyatakan bahwa *tools Burp Suite* memiliki keunggulan. Oleh karena itu, *tool Burp Suite* digunakan pada proses *scanning* metode *penetration testing* pada sistem PELAKAT.

Selanjutnya penelitian [16] membahas tentang *Penetration Testing* menggunakan teknik *SQL Injection* dan *XSS* (*Cross Site Scripting*) secara manual dengan cara memasukkan perintah SQL dan *script* XSS melalui input *form* yang bertujuan untuk analisis kerentanan keamanan dari *website* STIE Samarinda. Hasil penelitian menunjukkan adanya 14 kerentanan keamanan pada sistem tersebut. Terakhir penelitian [17] yang membahas pengujian *SQL Injection* menggunakan *android* yang memanfaatkan aplikasi *Termux* sebagai emulator terminal berbasis *Linux*. Hasil penelitian menyatakan adanya kerentanan keamanan pada *website* yang diuji.

Berdasarkan penjelasan diatas, penelitian ini akan membahas *penetration testing* teknik *SQL Injection* pada kerentanan keamanan sistem PELAKAT menggunakan *tools Burp Suite* dan *SQLMap*. Penelitian ini bertujuan untuk memastikan kualitas keamanan data pada sistem PELAKAT dari serangan.

2. METODE



Gambar 3. Tahapan *Penetration Testing*

Pengujian dilakukan dengan metode *penetration testing* teknik *SQL Injection*. Pengujian diimplementasikan pada sistem PELAKAT dan dilakukan *compare* pengujian pada sistem IFApps yang merupakan sistem informasi manajemen akademik teknik informatika Universitas Muhammadiyah Pontianak. *Compare* tersebut bertujuan untuk membandingkan hasil *penetration testing* sistem PELAKAT dan sistem IFApps. Pengujian ini difokuskan pada *form* input halaman login sistem. *Penetration testing* pada sistem PELAKAT dan *compare penetration testing* pada sistem IFApps dilakukan dalam lima tahapan seperti pada gambar 3 yang dimulai dari tahap *reconnaissance*, *scanning*, *vulnerability assessment*, *exploitation*, dan *reporting*.

2.1. Reconnaissance

Tahapan *reconnaissance* merupakan tahap analisis dan pengumpulan informasi yang berkaitan dengan sistem yang akan diuji. Analisis dan pengumpulan informasi ini dapat diperoleh

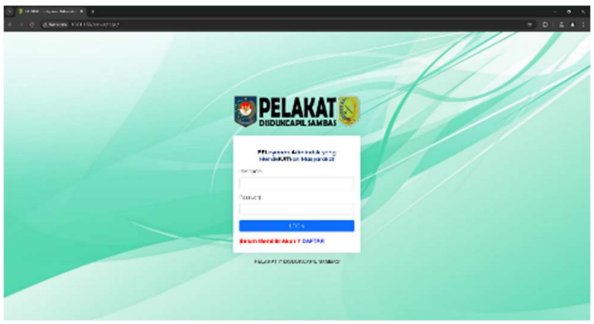
https://doi.org/10.25077/TEKNOSI.v11i1.2025.78-86

dari pihak pengembang sistem dan berfokus pada struktur sistemnya. Hal ini bertujuan untuk memudahkan proses *penetration testing* teknik *SQL Injection*. Adapun beberapa contoh informasi yang diperlukan pada sistem yang diuji yaitu *endpoint* yang akan diuji, parameter yang berpotensi rentan, jenis *framework*, *Back-End DBMS*, dan server *database* yang digunakan pada sistem.

2.2. *Scanning*

Tahap *scanning* merupakan tahapan pemindaian kerentanan pada sistem. *Scanning* bertujuan untuk mengidentifikasi dan mendeteksi kerentanan sistem terhadap *SQL Injection*. Pemindaian ini menggunakan *tool Burp Suite* yang difokuskan pada *form* halaman login untuk melakukan *intercept* pada HTTP *request* sistem.

Proses *scanning* dimulai dari pengaksesan halaman login sistem seperti pada gambar 4 yang merupakan pengaksesan halaman login sistem PELAKAT.



Gambar 4. Halaman Login Sistem PELAKAT

Kemudian memasukkan *value* pada *form input* login sistem yang nantinya HTTP *request* pada POST *method* login sistem yang dikirimkan ke server akan di *intercept* terlebih dahulu pada *Burp Suite*. Hal tersebut bertujuan untuk menganalisis respon server saat *value* parameter yang terdapat pada POST *method* login sistem diubah menjadi tanda petik tunggal. Jika teridentifikasi adanya kerentanan terhadap *SQL Injection*, maka respon server menampilkan pesan error SQL seperti pada gambar 5.

```
<b>
   Warning
</b>
:  mysqli_num_rows() expects parameter 1 to be
   mysqli_result, boolean given in <b>
   C:\xampp\htdocs\PELAKAT\login\dologin.php
</b>
  on line <b>
   12
</b>
```

Gambar 5. Contoh Pesan Error Pada Respon Server

Setelah penganalisisan selesai, HTTP *request* POST *method* login sistem disimpan menggunakan format *file* TXT yang nantinya akan dieksekusi dan dianalisis lebih lanjut pada tahap *exploitation* menggunakan *tool SQLMap*.

2.3. *Vulnerability Assessment*

Tahap *vulnerability assessment* merupakan penilaian kerentanan pada sistem yang diuji berdasarkan data informasi yang telah didapatkan pada tahap *reconnaissance* dan *scanning*. Penilaian kerentanan ini untuk mengidentifikasi potensi tingkat kerentanan sistem terhadap *SQL Injection* dan untuk mengetahui apakah kerentanan tersebut dapat dieksploitasi lebih lanjut.

Tabel 1. Rincian Tingkat Kerentanan Tahap *Reconnaissance*

Tingkat Kerentanan	Keterangan
Tinggi	*PHP Native* tanpa fitur keamanan memadai.
	Server fisik tanpa fitur keamanan memadai.
Sedang	*PHP Native* dilengkapi beberapa fitur keamanan.
	Server fisik yang dilengkapi keamanan seperti *firewall*.
Rendah	Menggunakan *framework*.
	Server berbasis *cloud*.

Berdasarkan Tabel 1, penilaian tingkat kerentanan pada tahap *reconnaissance* dibedakan menjadi tiga tingkatan yaitu tinggi, sedang, dan rendah. Penilaian tingkat kerentanan pada tahap *scanning* dibedakan menjadi tinggi dan rendah yang dapat dilihat pada tabel 2.

Tabel 2. Rincian Tingkat Kerentanan Tahap *Scanning*

Tingkat Kerentanan	Keterangan
Tinggi	Terdapat pesan error SQL pada respon server.
Rendah	Tidak ditemukan pesan error SQL pada respon server.

2.4. *Exploitation*

Tahap *exploitation* merupakan tahapan eksploitasi kerentanan pada sistem. Tahap ini bertujuan untuk memastikan kerentanan sistem terhadap *SQL Injection* berdasarkan hasil dari tahapan *reconnaissance* dan *scanning*. *Tool* yang digunakan untuk melakukan eksploitasi terhadap kerentanan sistem yaitu *SQLMap*. Eksploitasi ini mengeksekusi HTTP *request* POST *method* login sistem yang telah didapatkan pada tahap *scanning*.

```
PS C:\sqlmap> python sqlmap.py
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.8.7.2#dev}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: sqlmap.py [options]

sqlmap.py: error: missing a mandatory option (-d, -u, -l, -m, -r,
 -g, -c, --wizard, --shell, --update, --purge, --list-tampers or
--dependencies). Use -h for basic and -hh for advanced help
```

Gambar 6.Tampilan *SQLMap*

Gambar 6 merupakan tampilan *SQLMap* yang dijalankan melalui *CMD* (*Command Prompt*) menggunakan perintah "python sqlmap.py". Perintah tersebut melibatkan python karena *SQLMap* merupakan sebuah *tool* yang dikembangkan menggunakan bahasa pemrograman python. Eksploitasi kerentanan sistem menggunakan beberapa konfigurasi *command* dan disesuaikan dengan tingkat kerentanan sistem. Salah satu contoh konfigurasi *command* yang digunakan yaitu "python sqlmap.py -r request.txt --dbs --batch". Konfigurasi *command* tersebut bertujuan mengeksekusi *file* HTTP *request* POST *method* login sistem untuk mengakses dan menampilkan *database* sistem.

2.5. Reporting

Tahap *reporting* merupakan tahapan pelaporan berdasarkan analisis hasil pengujian *penetration testing* teknik *SQL Injection* pada sistem. Laporan ini mencakup kerentanan yang ditemukan pada sistem, tingkat kerentanan sistem, hasil eksploitasi kerentanan keamanan sistem, dan rekomendasi mitigasi pada sistem berdasarkan hasil pengujian.

3. HASIL

3.1. Penetration Testing

3.1.1. Hasil reconnaissance

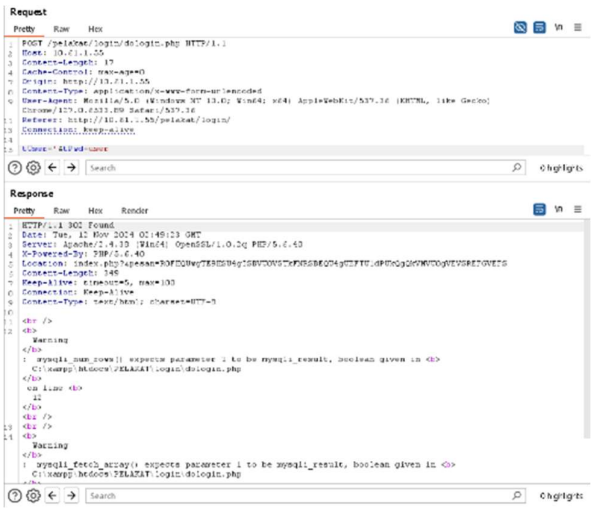
Hasil *reconnaissance* pada sistem PELAKAT merupakan pengumpulan informasi yang didapatkan dari pihak Disdukcapil Sambas selaku pengembang sistem PELAKAT dan analisis menggunakan *Burp Suite*. Rincian hasil *reconnaissance* tersebut dapat dilihat pada tabel 3.

Tabel 3. Hasil *Reconnaissance* Sistem PELAKAT

Hasil *Reconnaissance*	**Keterangan**
Endpoint yang diuji	/login.php
Parameter	username, password
Framework	Tanpa *framework* (*PHP Native*)
Back-End DBMS	*MySQL*
server *database*	Server fisik

3.1.2. Hasil Scanning

Intercept yang dilakukan pada HTTP *request* POST *method* login sistem PELAKAT mendapatkan dua parameter yang berpotensi rentan yaitu tUser dan tPwd.



Gambar 7. Respon Server Saat *Value* Parameter tUser Diubah

Berdasarkan pada gambar 7, respon server menampilkan pesan error SQL saat *value* parameter tUser diubah menjadi tanda petik tunggal. Oleh karena itu, sistem PELAKAT teridentifikasi mempunyai celah kerentanan keamanan terhadap *SQL Injection*. Untuk memastikan lebih lanjut kerentanan keamanan sistem PELAKAT terhadap *SQL Injection*, maka diperlukan eksploitasi menggunakan *tool SQLMap*.

3.1.3. Vulnerability Assessment

Pada tabel 4 merupakan penilaian kerentanan sistem PELAKAT terhadap *SQL Injection* yaitu hasil *reconnaissance* dan *scanning* memiliki tingkat kerentanan tinggi. Sehingga untuk memastikan hal tersebut, diperlukan tahap eksploitasi pada sistem PELAKAT.

Tabel 4. *Vulnerability Assessment* Sistem PELAKAT

Tahapan	**Tingkat Kerentanan**	**Hasil Identifikasi**
Hasil *Reconnaissance*	Tinggi	Tanpa *framework* (*PHP Native*)
		Server fisik
Hasil *Scanning*	.Tinggi	Pesan error pada respon server parameter tUser

3.1.4. Hasil Exploitation

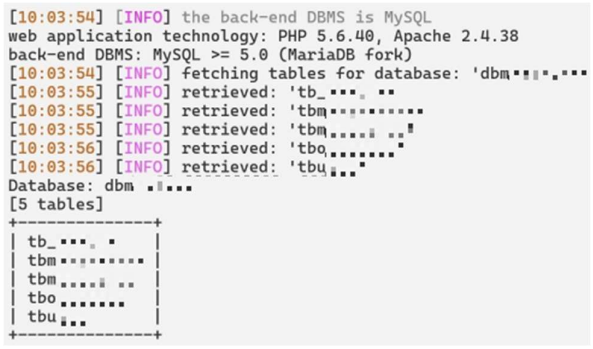
Eksploitasi yang pertama menggunakan konfigurasi *command* "python sqlmap.py -r request.txt --dbs –batch". Konfigurasi ini bertujuan untuk mengeksekusi *file* HTTP *request* POST *method* login sistem untuk mengakses dan menampilkan *database* sistem.

Gambar 8. Hasil Eksploitasi Pertama Sistem PELAKAT

Pada gambar 8 merupakan hasil eksploitasi pertama yang menunjukkan bahwa *SQLMap* mendeteksi parameter tUser pada POST *method* login sistem PELAKAT sebagai sebuah kerentanan dan hasilnya yaitu sistem PELAKAT mempunyai sebanyak sembilan *database* pada servernya.

Eksploitasi yang kedua menggunakan konfigurasi *command* "python sqlmap.py -r request.txt -D dbm******* --tables --batch". Konfigurasi ini bertujuan untuk mengakses dan menampilkan tabel yang ada di *database* sistem.

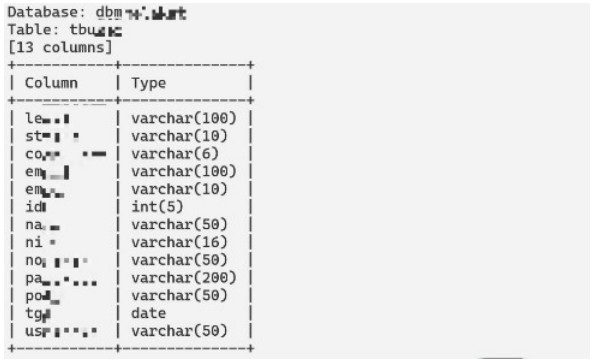
https://doi.org/10.25077/TEKNOSI.v11i1.2025.78-86



Gambar 9. Hasil Eksploitasi Kedua Sistem PELAKAT

Gambar 9 merupakan hasil eksploitasi kedua yang menunjukkan bahwa sistem PELAKAT mempunyai sebanyak lima tabel pada *database* dbm*******.

Eksploitasi yang ketiga menggunakan konfigurasi *command* "python sqlmap.py -r request.txt -D dbm******* -T tbu******* --columns --batch" yang bertujuan untuk mengakses dan menampilkan kolom yang ada di tabel *database* sistem.



Gambar 10. Hasil Eksploitasi Ketiga Sistem PELAKAT

Pada gambar 10 merupakan hasil eksploitasi ketiga yang menunjukkan bahwa sistem PELAKAT memiliki sebanyak 13 kolom pada tabel tbu******* yang diakses.

Eksploitasi yang keempat menggunakan konfigurasi *command* "python sqlmap.py -r request.txt -D dbm******* -T tbu******* -C em*******, na*******, ni******* --dump --batch" yang bertujuan untuk menampilkan dan mengekstrak beberapa data kolom menjadi file CSV.

Gambar 11. Hasil Eksploitasi Keempat Sistem PELAKAT

Gambar 11 merupakan hasil eksploitasi keempat untuk menampilkan tiga kolom yaitu em*******, na*******, ni******* dan secara otomatis *SQLMap* mengekspor data kolom tersebut sebagai *file* CSV.

3.1.5. Reporting

Berdasarkan hasil *penetration testing* teknik *SQL Injection* yang telah dilaksanakan pada sistem PELAKAT, tabel 5 merupakan rincian laporan hasil *penetration testing* yang menunjukkan bahwa sistem PELAKAT rentan terhadap *SQL Injection* dengan tingkat kerentanan yang tinggi.

Tabel 5. *Reporting Penetration Testing* Sistem PELAKAT

Bagian	**Keterangan**
Kerentanan yang ditemukan	Parameter tUser pada POST *method* login sistem PELAKAT rentan terhadap *SQL Injection*.
Tingkat kerentanan	Tinggi.
Hasil Eksploitasi	9 *database*
	5 tabel pada *database* dbm*******
	13 kolom pada tabel tbu*******
Rekomendasi mitigasi	Menerapkan *WAF* (*Web Application Firewall*).
	Melakukan validasi input pada *form* input yang bertujuan untuk memproses input sesuai kriteria yang ditentukan.
	Menggunakan *prepared statements* yang bertujuan untuk memisahkan *value* input dengan *query* SQL.
	Menggunakan *framework* seperti Laravel.
	Menggunakan penyimpanan *database* berbasis *cloud*.

3.2. Compare Penetration Testing

3.2.1. Hasil reconnaissance

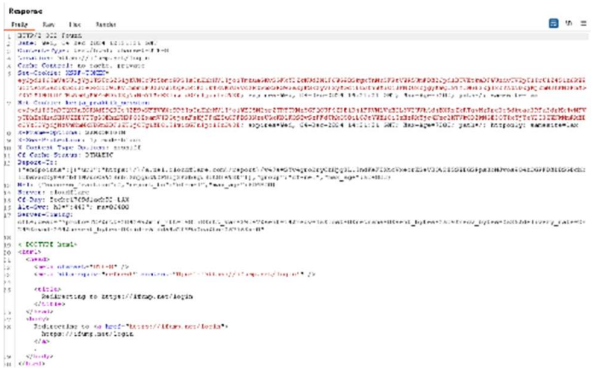
Pada tabel 6 merupakan rincian hasil *reconnaissance* pada sistem IFApps.

Tabel 6. Hasil *Reconnaissance* Sistem IFApps

Hasil *Reconnaissance*	**Keterangan**
Endpoint yang diuji	/login.php
Parameter	username, password
Framework	Tanpa *framework* (*PHP Native*)
Back-End DBMS	*MySQL*
server *database*	Server *cloud*

3.2.2. Hasil Scanning

Intercept yang dilakukan pada HTTP *request* POST *method* login sistem PELAKAT mendapatkan tiga parameter yang berpotensi rentan yaitu _token, email, dan password.



Gambar 12. Respon Server Saat *Value* Parameter Email Diubah

Ketika *value* parameter _token, email, dan password diubah menjadi tanda petik tunggal, respon server tiap parameter tidak menampilkan pesan error SQL seperti yang terlihat pada gambar 12. Untuk memastikan lebih lanjut kerentanan keamanan sistem IFApps terhadap *SQL Injection*, maka diperlukan eksploitasi menggunakan *tool SQLMap*.

3.2.3. *Vulnerability Assessment*

Pada tabel 7 merupakan penilaian kerentanan sistem IFApps terhadap *SQL Injection* yaitu hasil *reconnaissance* dan *scanning* memiliki tingkat kerentanan rendah. Sehingga untuk memastikan hal tersebut, diperlukan tahap eksploitasi pada sistem IFApps.

Tabel 7. *Vulnerability Assessment* Sistem IFApps

Tahapan	Tingkat Kerentanan	Hasil Identifikasi
Hasil *Reconnaissance*	Rendah	*framework* Laravel
		Server *cloud*
Hasil *Scanning*	.Rendah	Tidak ada pesan error SQL pada respon server

3.2.4. *Hasil Exploitation*

Eksploitasi pertama menggunakan konfigurasi *command* "python sqlmap.py -r request.txt --dbs –batch". Konfigurasi ini bertujuan untuk mengeksekusi *file* HTTP *request* POST *method* login sistem untuk mengakses dan menampilkan *database* sistem.

```
[09:40:42] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[09:40:43] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[09:40:44] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[09:40:44] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other
(potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[09:40:45] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:40:46] [WARNING] POST parameter 'email' does not seem to be injectable
[09:40:46] [WARNING] POST parameter 'password' does not appear to be dynamic
[09:40:46] [WARNING] heuristic (basic) test shows that POST parameter 'password' might
not be injectable
[09:40:47] [INFO] testing for SQL injection on POST parameter 'password'
[09:40:47] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:40:48] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:40:48] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GR
OUP BY clause (EXTRACTVALUE)'
[09:40:49] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[09:40:50] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVIN
G clause (IN)'
[09:40:51] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[09:40:52] [INFO] testing 'Generic inline queries'
[09:40:52] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[09:40:52] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[09:40:53] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)
'
[09:40:53] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[09:40:54] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[09:40:55] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[09:40:56] [INFO] testing 'Oracle AND time-based blind'
[09:40:56] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:40:58] [WARNING] POST parameter 'password' does not seem to be injectable
[09:40:58] [CRITICAL] all tested parameters do not appear to be injectable. Try to incr
ease values for '--level'/'--risk' options if you wish to perform more tests. If you su
spect that there is some kind of protection mechanism involved (e.g. WAF) maybe you cou
ld try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random
-agent'
[09:40:58] [WARNING] HTTP error codes detected during run:
419 (?) - 144 times
```

Gambar 13. Hasil Eksploitasi Pertama Sistem IFApps

Pada gambar 13 merupakan hasil eksploitasi pertama pada sistem IFApps yang menyatakan bahwa semua parameter pada sistem aman terhadap *SQL Injection*.

Eksploitasi kedua menggunakan konfigurasi *command* "python sqlmap.py -r request.txt --level=5 --risk=2 --dbs --batch". Konfigurasi ini menggunakan level untuk menentukan jumlah payload yang akan diuji dan risiko untuk menentukan tingkat risiko dari payload yang akan diuji.

```
[10:14:47] [INFO] POST parameter 'email' appears to be 'Oracle AND time-based blind (he
avy query)' injectable
it looks like the back-end DBMS is 'Oracle'. Do you want to skip test payloads specific
 for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'Oracle' extending provid
ed risk (2) value? [Y/n] Y
[10:14:47] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[10:14:47] [INFO] automatically extending ranges for UNION query injection technique te
sts as there is at least one other (potential) technique found
[10:14:48] [INFO] testing 'Generic UNION query (random number) - 1 to 20 columns'
[10:14:48] [INFO] testing 'Generic UNION query (NULL) - 21 to 40 columns'
[10:14:52] [INFO] testing 'Generic UNION query (random number) - 21 to 40 columns'
[10:14:55] [INFO] testing 'Generic UNION query (NULL) - 41 to 60 columns'
[10:15:00] [INFO] testing 'Generic UNION query (random number) - 41 to 60 columns'
[10:15:04] [INFO] testing 'Generic UNION query (NULL) - 61 to 80 columns'
[10:15:07] [INFO] testing 'Generic UNION query (random number) - 61 to 80 columns'
[10:15:10] [INFO] testing 'Generic UNION query (NULL) - 81 to 100 columns'
[10:15:14] [INFO] testing 'Generic UNION query (random number) - 81 to 100 columns'
[10:15:19] [INFO] checking if the injection point on POST parameter 'email' is a false
positive
[10:15:19] [WARNING] false positive or unexploitable injection point detected
[10:15:19] [WARNING] POST parameter 'email' does not seem to be injectable
```

Gambar 14. Hasil Eksploitasi Kedua Sistem IFApps

Gambar 14 merupakan hasil eksploitasi kedua yang menunjukkan bahwa *SQLMap* mendeteksi *back-end database* pada sistem yang mirip seperti *Oracle* yang dideteksi melalui parameter email. Walaupun *SQLMap* berhasil mendeteksi kemiripan *back-end database* pada sistem, parameter email dan yang lainnya menyatakan tidak rentan terhadap *SQL Injection*.

Eksploitasi ketiga menggunakan konfigurasi *command* "python sqlmap.py -r request.txt --level=5 --risk=2 –tamper= "randomcase, space2comment" –dbs --batch". Konfigurasi ini menggunakan dua *script tamper* untuk menghindari *input filter*.

```
[09:57:47] [INFO] testing for SQL injection on parameter 'Referer'
[09:57:47] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:58:01] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (subquery -
 comment)'
[09:58:12] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (comment)'
[09:58:15] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:58:15] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL)'
[09:58:15] [INFO] testing 'Boolean-based blind - Parameter replace (DUAL - original val
ue)'
[09:58:15] [INFO] testing 'Boolean-based blind - Parameter replace (CASE)'
[09:58:16] [INFO] testing 'Boolean-based blind - Parameter replace (CASE - original val
ue)'
[09:58:16] [INFO] testing 'HAVING boolean-based blind - WHERE, GROUP BY clause'
[09:58:26] [INFO] testing 'Generic inline queries'
[09:58:27] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:58:43] [INFO] testing 'Generic UNION query (random number) - 1 to 10 columns'
[09:59:00] [WARNING] parameter 'Referer' does not seem to be injectable
[09:59:00] [CRITICAL] all tested parameters do not appear to be injectable. Try to incr
ease values for '--level'/'--risk' options if you wish to perform more tests
[09:59:00] [WARNING] HTTP error codes detected during run:
419 (?) - 4240 times
```

Gambar 15. Hasil Eksploitasi Ketiga Sistem IFApps

Pada gambar 15 merupakan hasil eksploitasi ketiga pada sistem IFApps yang menunjukkan bahwa semua parameter yang telah diuji tidak menunjukkan adanya kerentanan terhadap *SQL Injection.* Berdasarkan hasil dari ketiga konfigurasi *command* yang dijalankan *SQLMap*, sistem IFApps memiliki kerentanan yang rendah terhadap *SQL Injection*.

3.2.5. *Reporting*

Berdasarkan hasil *compare penetration testing* teknik *SQL Injection* yang telah diimplementasikan pada sistem IFApps, tabel 8 merupakan rincian laporan hasil *compare penetration testing* yang menunjukkan bahwa sistem IFApps memiliki tingkat kerentanan yang rendah terhadap *SQL Injection*. Oleh karena itu, kecil kemungkinan *database* sistem dapat diakses oleh pihak

https://doi.org/10.25077/TEKNOSI.v11i1.2025.78-86

yang tidak bertanggung jawab. Hasil eksploitasi menggunakan *SQLMap* hanya menunjukkan kemiripan *back-end database* yang terlihat seperti *Oracle*. Namun untuk tetap menjaga keamanan sistem, *update* pada keamanan sistem tetap diperlukan sesuai dengan perkembangan sehingga dapat meminimalisir kerentanan lainnya.

Tabel 8. *Reporting Compare Penetration Testing* Sistem IFApps

Bagian	Keterangan
Kerentanan yang ditemukan	Tidak ditemukan kerentanan terhadap *SQL Injection*.
Tingkat kerentanan	Rendah.
Hasil Eksploitasi	Kemiripan *back-end database* yang terlihat seperti *Oracle*.
Rekomendasi mitigasi	Rutin melakukan *update* keamanan sistem.

4. PEMBAHASAN

Berdasarkan hasil *penetration testing* teknik *SQL Injection* pada sistem PELAKAT, diketahui bahwa sistem PELAKAT memiliki tingkat kerentanan yang tinggi terhadap *SQL Injection*. Hal tersebut diketahui melalui *intercept* HTTP *request* POST *method* login sistem PELAKAT, karena respon server menampilkan pesan error SQL saat *value* parameter tUser diubah menjadi tanda petik tunggal. Selain itu, eksploitasi menggunakan *SQLMap* pada sistem PELAKAT menyatakan bahwa *database* sistem dapat diakses secara ilegal. *Database* yang teridentifikasi oleh *SQLMap* yaitu sebanyak sembilan *database*, lima tabel pada *database* yang dituju, dan terdapat 13 kolom pada tabel yang dituju.
Hasil *compare* pada sistem IFApps menunjukkan bahwa sistem IFApps memiliki tingkat kerentanan yang rendah terhadap *SQL Injection*. Hal tersebut diketahui melalui *penetration testing* yang dilakukan pada sistem IFApps. Respon server tiap parameter tidak menampilkan pesan error SQL saat *value* tiap parameter diubah menjadi tanda petik tunggal pada *intercept* HTTP *request* POST *method* login sistem IFApps. Selain itu, hasil eksploitasi tidak dapat mengakses *database* sistem meskipun *SQLMap* menemukan kemiripan *back-end database* sistem yang terlihat seperti *Oracle*. Hal tersebut dikarenakan sistem IFApps telah dikembangkan dengan menerapkan beberapa fitur keamanan sehingga dapat meminimalisir kerentanan pada sistemnya.

Sistem IFApps dikembangkan menggunakan *framework* Laravel yang telah menyediakan fitur keamanan bawaan dan server *database* sistem IFApps berbasis *cloud*. Hal tersebut dapat diterapkan pada sistem PELAKAT sebagai rekomendasi mitigasi. Selain itu, sistem PELAKAT dapat menerapkan *WAF*, validasi input, dan *prepared statements*. Beberapa rekomendasi mitigasi tersebut bertujuan untuk meningkatkan keamanan sistem PELAKAT terhadap *SQL Injection*.

5. KESIMPULAN

Berdasarkan penelitian yang telah dilaksanakan, *penetration testing* pada keamanan sistem PELAKAT menggunakan teknik *SQL Injection* dilakukan dalam lima tahapan yaitu *reconnaissance*, *scanning*, *vulnerability assessment*, *exploitation*, dan *reporting*. Pada tahap *scanning* menggunakan *tool Burp Suite* bertujuan untuk melakukan *intercept* HTTP *request* pada POST *method* login sistem dan untuk mengetahui respon server sistem jika *value* parameter diubah menjadi tanda petik tunggal. Hasil *scanning* pada sistem PELAKAT menunjukkan bahwa sistem PELAKAT teridentifikasi memiliki kerentanan terhadap *SQL Injection*. Hal tersebut diketahui karena respon server menunjukkan adanya pesan error SQL saat *value* parameter tUser diubah menjadi tanda petik tunggal. Pada tahap *exploitation* menggunakan *tool SQLMap* yang bertujuan untuk mengeksploitasi lebih lanjut kerentanan sistem PELAKAT terhadap *SQL Injection*, didapatkan hasil yang menunjukkan bahwa *SQLMap* dapat mengidentifikasi *database* pada sistem PELAKAT.

Penetration testing pada keamanan sistem PELAKAT menggunakan teknik *SQL Injection* dinyatakan berhasil mengidentifikasi kerentanan keamanan sistemnya. Sehingga diperlukan beberapa rekomendasi mitigasi seperti menerapkan *WAF* (*Web Application Firewall*), melakukan validasi input pada *form* input, menggunakan *prepared statements*, menggunakan *framework* seperti Laravel, dan menggunakan penyimpanan *database* berbasis *cloud*.

DAFTAR PUSTAKA

[1] G. Suprianto, "Penetration Testing Pada Sistem Informasi Jabatan Universitas Hayam Wuruk Perbanas," *InComTech J. Telekomun. dan Komput.*, vol. 12, no. 2, p. 129, 2022, doi: 10.22441/incomtech.v12i2.15093.

[2] Zulkifli and Samsir, "Implementasi Sistem Keamanan SQL Injection Dalam berbasis web," *U-NET J. Tek. Inform.*, vol. 4, no. 1, pp. 8–13, 2020, doi: 10.52332/u-net.v4i1.164.

[3] F. Gurcan, G. G. M. Dalveren, N. E. Cagiltay, D. Roman, and A. Soylu, "Evolution of Software Testing Strategies and Trends: Semantic Content Analysis of Software Research Corpus of the Last 40 Years," *IEEE Access*, vol. 10, no. September, pp. 106093–106109, 2022, doi: 10.1109/ACCESS.2022.3211949.

[4] D. P. Putranto, J. Jayanta, and B. Hananto, "Analisis Keamanan Website Leads UPNVJ Terhadap Serangan SQL Injection & Sniffing Attack," *Inform. J. Ilmu Komput.*, vol. 18, no. 3, p. 230, 2022, doi: 10.52958/iftk.v18i3.4690.

[5] OWASP, "OWASP Top Ten Project," *Www*, 2010. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project []. (accessed Jan. 03, 2025).

[6] A. B. Musa *et al.*, "Perancangan Sistem Informasi Data Kependudukan Berbasis Web Pada Kelurahan Lokoboko Kecamatan Ndona," vol. 3, no. 1, pp. 34–40, 2022.

[7] D. Galin, *Software Quality Assurance From theory to implementation Software Quality Assurance From theory to implementation CYAN MAGENTA YELLOW BLACK*. 2004. [Online]. Available: www.pearsoned.co.uk [Dec. 27, 2024].

[8] G. O'Regan, *Concise Guide to Software Testing*. London: Springer Nature, 2019.

[9] S. W. Ningsih, A. Almaarif, and A. Widjadjarto, "Vulnerability Testing Analysis of XYZ Regional Government Site Using PTES," *JATISI (Jurnal Tek. Inform. dan Sist. Informasi)*, vol. 8, no. 3, pp. 1543–1556, 2021, doi: 10.35957/jatisi.v8i3.1224.

[10] T. I. M. Pratama, M. D. F. Songida, and I. Gunawan, "Analisis Serangan dan Keamanan pada SQL Injection: Sebuah Review Sistematik," *JIIFKOM (Jurnal Ilm.*

Inform. dan Komputer), vol. 1, no. 2, pp. 27–32, 2022, doi: 10.51901/jiifkom.v1i2.230.

[11] N. Albalawi, N. Alamrani, R. Aloufi, M. Albalawi, A. Aljaedi, and A. R. Alharbi, "The Reality of Internet Infrastructure and Services Defacement: A Second Look at Characterizing Web-Based Vulnerabilities," *Electron.*, vol. 12, no. 12, 2023, doi: 10.3390/electronics12122664.

[12] H. Herman, I. Riadi, and Y. Kurniawan, "Vulnerability Detection With K-Nearest Neighbor and Naïve Bayes Method using Machine Learning," *Int. J. Artif. Intell. Res.*, vol. 7, no. 1, p. 10, 2023, doi: 10.29099/ijair.v7i1.795.

[13] A. Alanda, D. Satria, M. I. Ardhana, A. A. Dahlan, and H. A. Mooduto, "Web application penetration testing using sql injection attack," *Int. J. Informatics Vis.*, vol. 5, no. 3, pp. 320–326, 2021, doi: 10.30630/joiv.5.3.470.

[14] P. G. S. Adinata, I. P. W. P. Putra, N. P. A. I. Juliantari, and K. D. A. Sutrisna, "Analisis Perbandingan Tools SQL Injection Menggunakan SQLmap, SQLsus dan The Mole," *Inform. J. Ilmu Komput.*, vol. 18, no. 3, p. 286, 2022, doi: 10.52958/iftk.v18i3.5373.

[15] Ni Putu Ana Rainita, Anak Agung Istri Callysta Athalia, Made Diva Putera Ananta, I Ketut Pratista Tri Pramana, Gede Arna Jude Saskara, and I Made Edy Listartha, "Analisis Perbandingan Vulnerability Scanning Pada Website Dvwa Menggunakan Owasp Nikto Dan Burpsuite," *J. Inform. Dan Tekonologi Komput.*, vol. 3, no. 2, pp. 89–97, 2023, doi: 10.55606/jitek.v3i2.908.

[16] T. Anugrah, "Penetration Testing Keamanan Website Stie Samarinda Menggunakan Teknik Sql Injection Dan Xss," *J. Inform. dan Tek. Elektro Terap.*, vol. 12, no. 1, pp. 618–624, 2024, doi: 10.23960/jitet.v12i1.3882.

[17] A. Andria and R. Pamungkas, "Penetration Testing Database Menggunakan Metode SQL Injection Via SQLMap di Termux," *Indonesian Journal of Applied Informatics*, vol. 5, no. 1. p. 1, 2021. doi: 10.20961/ijai.v5i1.40845.