BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Sebelumnya

Mi-Yeon Kim menyebutkan pada penelitian yang berjudul *data mining based SQL injection attack detection using internal query trees* bahwa pada penelitian tersebut menggabungkan metode mengubah pohon permintaan menjadi vektor n-dimensi dengan menggunakan menggunakan urutan multi dimensi sebagai representasi [11]. Kemudian metode mengekstrak fitur sintaksis, serta fitur semantik untuk saat membuat fitur vektor. Selain itu terdapat metode mengubah nilai *string* menjadi nilai numerik dengan menggabungkan beberapa model statistik. Model gabungan peta satu nilai *string* untuk satu nilai numerik mengandung beberapa karakteristik masing-masing nilai *string*. Mendeteksi serangan *SQL injection* pada tingkat *database* berdasarkan data *mining* bukan berdasarkan data yang diambil dari *log server*. Pada percobaan dari penelitian tersebut menunjukan bahwa penelitian tersebut secara signifikan meningkatkan kemungkinan mendeteksi dengan benar serangan *SQL injection*.

Pada penelitian yang dibuat oleh Ain Zubaidin Mohd Saleh, dkk menyebutkan bahwa pada penelitian ini mengembangkan metode deteksi kerentanan aplikasi web dengan menggunakan *Boyer-Moore String Matching Algorithm* [12]. Algoritma tersebut akan memindai karakter demi karakter dari kanan ke kiri sampai kecocokan ditemukan. Pada percobaan yang dilakukan menghasilkan keakuratan untuk mendeteksi kerentanan berdasarkan *false negative* dan tidak memiliki *false positive* dengan pengolahan yang rendah. Teknik ini diterapkan untuk mendeteksi kerentanan seperti *SQL injection*, XSS, *buffer overflow*, CSRF. Penelitian ini memiliki kekurangan karena ia mendeteksi berdasarkan pencocokan dengan cara memindai, apabila tidak ditemukan kecocokan dianggap tidak terjadi serangan.

Pada penelitian yang dibuat oleh Anamika Joshi dab Gettha menggunakan metode klasifikasi *naïve bayes* [13]. Ketika pengguna memasukan *query* SQL dari

URL situs web, sistem monitor melakukan pengecekan permintaan SQL, memecahnya menjadi jumlah kata kunci berdasarkan ruang kosong dalam *query* dinamis dan menghitung panjang *query* SQL dinamis. Selain itu juga menghitung jumlah kata kunci yang ada dalam *query* tersebut dan mengirimkan nilai numerik panjang dan kata kunci *query* dinamis ke *classifier*. *Classifier* kemudian menghitung probabilitas injeksi SQL dalam *query* dinamis berdasarkan hasil yang diterima oleh konverter, dan kemudian membandingkan probabilitas injeksi SQL yang dihitung dengan yang ditentukan oleh ambang pengguna sebagai kumpulan data pelatihan yang membantu dalam menghitung probabilitas permintaan dan probabilitas yang sah dari *query* jahat. Jika probabilitas *query* SQL dinamis dihitung oleh *classifier* probabilitas yang cocok dari *query* yang sah dalam *dataset* pelatihan, *query* diperbolehkan. Jika tidak maka itu diblokir. Kekurangan pada penelitian ini masih menggunakan *dataset* yang kecil dan belum bisa melakukan pencegahan serangan *blind sql injection.*

Penilitian yang dibuat oleh Rahajeng Ellysa membahas metode pengamanan aplikasi web untuk deteksi *SQL injection* menggunakan algoritma SQL-IF [14]. Sistem ini berindak seperti *proxy* yang dapat difungsikan untuk melakukan penyaringan masukan ketika menjalankan sebuah perintah pada aplikasi web. Apabila terdapat serangan *SQL injection*, sistem ini mampu mendeteksi dan mencegah dengan cara menambahkan karakter yang tidak berbahaya. Sehingga *attacker* tidak dapat melakukan serangan kepada aplikasi web tersebut. Sistem ini mampu melakukan penyaringan parameter injeksi seperti {([„„&+=<><= >=])} serta kata kunci SQL seperti *union, select,intersect, insert, update, delete, drop, truncate*, serta karakter *boolean („ or „„|"or"|"AND"|"and")*. Jika *field* berisi suatu parameter yang berbahaya maka akan dideteksi sebagai serangan. Berdasarkan hasil uji coba, sistem ini mampu mendeteksi masukan dari pengguna sehingga serangan dapat dicegah. Sistem ini belum teruji apabila parameter injeksi dilakukan *encoding.*

Penelitian yang dibuat oleh Tejinderdeep Singh Kalsi menjelaskan metode untuk mendeteksi dan mencegah serangan *SQL Injection* menggunakan algoritma *efferent* dengan cara mencocokan pola [15]. Dibedakan menjadi dua fase yang

terdiri dari fase statis dan fase dinamis. Dalam daftar pola statis menggunakan anomali. Pada fase statis, pengguna yang melakukan masukan dengan SQL akan dicek menggunakan algortima pencocokan. Kemudian pada fase dinamis, jika terjadi bentuk anomali yang baru akan menunjukan tanda bahaya dan pola anomali baru akan dihasilkan. Pola anomali baru akan dimasukkan ke daftar pola statik. Teknik campuran yang disarankan akan dilakukan dalam dua tahap utama: analisis dinamis dan analisis statis. Tahap pertama adalah metode analisis dinamis bergantung pada penerapan metode pelacakan untuk memproses dan memantau proses pelaksanaan semua pertanyaan yang diterima. Hasil dari objek yang terdeteksi ini akan dicocokkan dengan sekumpulan perubahan yang direncanakan. Jika terdapat serangan *SQL injection* diteruskan ke tahap berikutnya. Tahap berikutnya adalah tahap analisis statis yang melakukan pencocokan *string* antara query SQL yang diterima dan *query* SQL yang diharapkan sebelumnya untuk menghentikan segala *query* berbahaya.

Pada penelitian yang dibuat oleh Dr. M. Amutha Prabakar, dkk ini menggunakan algotirma *pattern matching* untuk mendeteksi dan mencegah serangan *sql injection* [16]. Paramater yang masuk akan dicocokan dengan *pattern* yang telah didaftarkan pada sistem. Apabila cocok maka permintaan akan dilakukan pemblokiran. Apabila tidak ada yang cocok dengan *pattern* maka permintaan diperbolehkan dan diteruskan. Cara ini bisa dapat dilewati dengan cara melakukan *encoding* parameter yang akan dikirimkan.

Tabel 2.1 Perbandingan dengan Penelitian Sebelumnya

No	Judul Penelitian	Ringkasan	Perbedaan
1	*Data-mining based SQL injection attack detection using internal query trees.*	Menggunakan klasifikasi SVM (*Support Vector Machine)* dengan *query trees.* Algoritma SVM yang digunakan perlu	Pada penelitian ini, peneliti akan menggunakan *naïve bayes* untuk metode klasifikasi. Deteksi serangan tidak berada

No	Judul Penelitian	Ringkasan	Perbedaan
		adanya *log database* untuk membantu deteksi suatu serangan *injection.* Mendeteksi serangan *SQL injection* pada tingkat *database* berdasarkan data *mining* bukan berdasarkan data yang diambil dari *log server*.	pada tingkat *database*, namun berdasarkan parameter yang dimasukkan dan dikirimkan.
2	*A Method for Web Application Vulnerabilities Detection by Using Boyer-Moore String Matching Algorithm.*	Penelitian ini mengembangkan metode deteksi kerentanan aplikasi web dengan menggunakan *Boyer-Moore String Matching Algorithm.* Algoritma tersebut akan memindai karakter demi karakter dari kanan ke kiri sampai kecocokan ditemukan.	Deteksi yang dilakukan menggunakan algoritma *SQL Injection Free Secure* yaitu dengan mencocokan parameter yang dimasukkan oleh pengguna dengan dengan kata kunci yang telah didaftarkan pada sistem.

No	Judul Penelitian	Ringkasan	Perbedaan
3	*SQL Injection detection using machine learning.*	Menggunakan metode *naïve bayes* untuk melakukan deteksi dan pencegahan serangan. Membutuhkan data serangan dan data normal dan kemudian melakukan *training data.*	Penelitian yang akan dibuat oleh peneliti juga menggunakan metode klasifikasi *naïve bayes*, namun yang membedakan adalah peneliti mengkombinasikan metode ini dengan algoritma SQL-IF. Apabila serangan tidak bisa diatasi menggunakan algortima SQL-IF, kemudian *naïve bayes* akan bekerja untuk mendeteksi dan mencegahnya.
4	Pendeteksi Serangan SQL Injection Menggunakan Algoritma SQL *Injection Free Secure* pada Aplikasi Web.	Menggunakan algoritma SQL-IF. Sistem ini berindak seperti *proxy* yang dapat difungsikan untuk melakukan penyaringan masukan ketika menjalankan sebuah perintah	Penelitian yang akan dibuat oleh peneliti juga menggunakan algoritma SQL-IF namun yang membedakan adalah peneliti mengkombinasikan algoritma SQL-IF dengan metode

No	Judul Penelitian	Ringkasan	Perbedaan
		pada aplikasi web. Apabila terdapat serangan *SQL injection*, sistem ini mampu mendeteksi dan mencegah dengan cara menambahkan karakter yang tidak berbahaya.	klasifikasi *naïve bayes* untuk mendapatkan tingkat keakuratan deteksi serangan.
5	*An Efficient Technique For Preventing Sql Injection Attack Using Pattern Matching Algorithm.*	Menggunakan metode *pattern matching algorithm.* Solusi yang ditawarkan adalah melakukan pencocokan antara permintaan yang masuk dengan *pattern.* Apabila cocok, permintaan diblokir. Apabila tidak, permintaan diperbolehkan.	Penelitian ini menggunakan kombinasi dua metode antara klasifikasi *naïve bayes* dan algortima SQL-IF. Algoritma SQL-IF hampir sama dengan *pattern matching*, namun pada SQL-IF permintaan yang mengandung parameter berbahaya akan diganti dengan parameter yang tidak berbahaya.

Berdasarkan penelitian-penelitian diatas, terdapat beberapa cara untuk melakukan deteksi dan mencegah serangan ini. Penelitian menggunakan *machine*

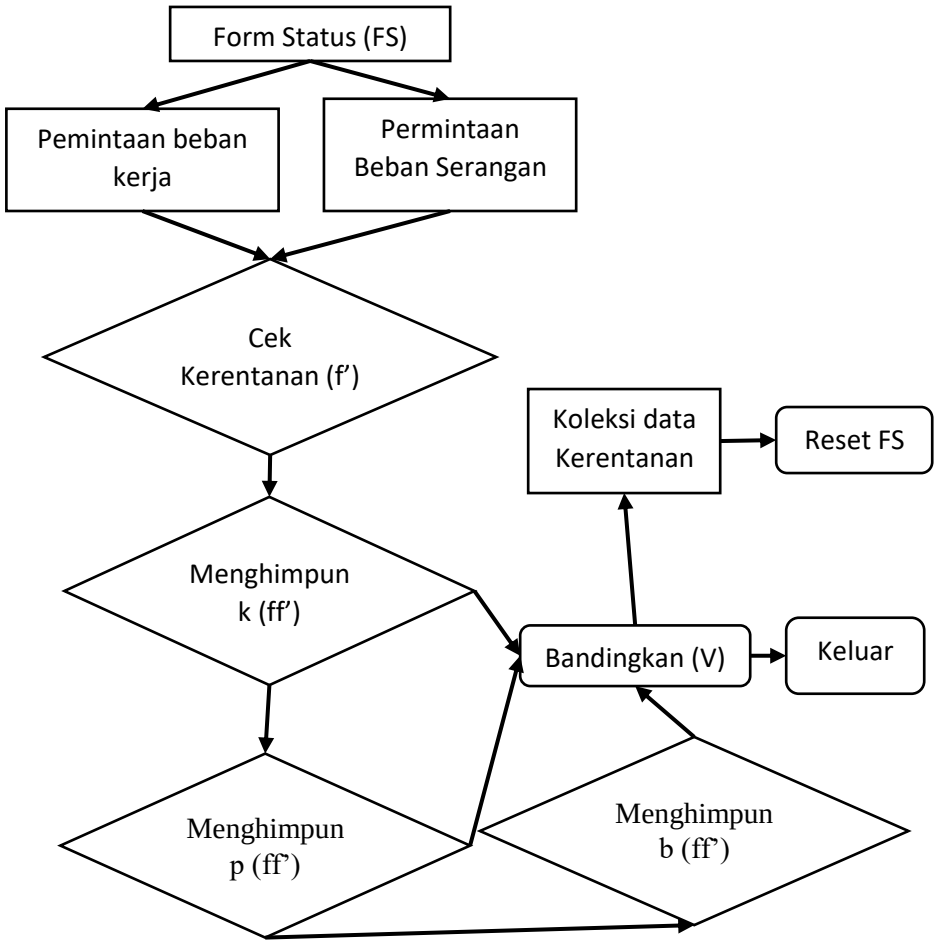
learning dengan cara klasifikasi menggunakan metode *naïve bayes* mampu melakukan deteksi dengan akurasi 93.3%. Sedangkan peneltian menggunakan metode algoritma *SQL injection free secure* dapat melakukan deteksi dalam waktu 5 ms. Namun pada penelitian tersebut masih mempunyai kekurangan, pada penelitian *machine learning* dengan cara klasifikasi menggunakan metode *naïve bayes* mempunyai kekurangan belum bisa mendeteksi serangan *blind SQL injection.* Sedangkan pada penelitian menggunakan algoritma *SQL injection free secure* masih belum teruji apabila serangan oleh *attacker* dilakukan *encoding.* Oleh sebab itu pada penelitian ini akan mencoba melakukan kombinasi antara algoritma *SQL injection free secure* dan metode klasifikasi *naïve bayes* untuk menguji keakuratan dan keefisienan dalam mendeteksi dan mencegah serangan *SQL injection.*

2.2 Landasan Teori

2.2.1 Algoritma *SQL Injection Free Secure*

Algoritma *SQL Injection Free Secure* (SQL-IF) merupakan algoritma yang menggunakan metode dinamis berfokus untuk peningkatan serangan IF (*Injection Free*), berbagai jenis tes dilakukan untuk mendeteksi dan mencegah terjadinya serangan SQL injection. Algoritma SQL-IF dijelaskan pada diagram alir Gambar 2.1 [17].

Berdasarkan alur diagaram dari gambar, apabila terdapat masukan dari pengguna baik masukan serangan atau bukan maka akan dicek terlebih dahulu. Kemudian menghimpun koleksi *keyword* yang tersimpan pada sistem seperti *union, select, all, tables, insert, update, create, information_schema, database, order* dan kata kunci lainnya. Setelah itu menghimpun koleksi karakter spesial seperti ‘, “, (,), --, -+, #, <,>,=,[,],{,} dan karater spesial lainnya. Kemudian menghimpun koleksi *bolean* seperti *AND, OR, XOR, or, and, xor.* Setelah itu masukan akan dibandingkan dengan koleksi yang sudah dihimpun. Apabila terdapat parameter yang cocok dengan koleksi maka disebut sebagai serangan. Kemudian parameter tersebut akan diubah menjadi parameter yang tidak berbahaya dan diteruskan pada *server.*



Gambar 2.1 Gambar Diagram Alir Algoritma SQL-IF [17].

Sebelum melakukan serangan *SQL injection* biasanya *attacker* mengirimkan parameter petik (‘) untuk melakukan *test* apakah aplikasi web tersebut rentan terhadap serangan *SQL injection* atau tidak. Apabila aplikasi web tersebut rentan, maka akan mengembalikan pesan *error*. Setelah itu *attacker* akan menggunakan *keyword SQL* untuk mendapatkan data yang diinginkan, seperti *order by, union* dan *select.* Logika operator seperti *OR* juga sering digunakan oleh *attacker* untuk melakukan *bypassing authentication* pada halaman *login.*

Algoritma *SQL-IF* digunakan untuk mendeteksi masukan-masukan seperti kasus diatas seperti mendeteksi petik, *keyword sql* dan logika operator. Harapannya apabila terdapat parameter yang mengandung serangan dapat dicegah dengan algoritma ini. Sehingga *attacker* tidak dapat melakukan eksploitasi serangan *SQL injection.*

2.2.2 Metode Klasifikasi *Naïve Bayes*

Metode *naïve bayes* merupakan metode klasifikasi probabilistik yang digunakan untuk menghitung sekumpulan probabilitas dengan cara menjumlahkan frekuensi dan kombinasi nilai dari dataset yang disediakan. Metode *naïve bayes* dapat digunakan untuk memprediksi masa yang akan datang berdasarkan pengalaman sebelumnya. Keuntungan menggunakan metode ini yaitu metode ini tidak membutuhkan *data training* yang besar untuk menentukan estimasi parameter dalam proses pengklasifikasian [18]. Persamaan teorema *Bayes* adalah sebagai berikut :

$$P(Y|X) = \frac{P(X|Y).P(Y)}{P(X)} \quad (2.1)$$

X :Data dengan *class* yang belum diketahui.
Y : Hipotesis data yang merupakan suatu *class* spesifik.
P(Y|X) :Probabilitas hipotesis Y berdasar kondisi X (posteriori probabilitas).
P(Y) : Probabilitas hipotesis Y (prior probabilitas).
P(X|Y) :Probabilitas X berdasarkan kondisi pada hipotesis Y.
P(X) : Probabilitas dari X.

Proses klasifikasi pada metode *naïve bayes* memerluhkan petunjuk untuk menentukan kelas yang cocok untuk sampel yang dianalisis. Metode *naïve bayes* dapat disesuaikan menjadi sebagai berikut :

$$P(C|F1 \dots Fn) = \frac{P(C)P(F1 \dots Fn|C)}{P(F1 \dots Fn)} \quad (2.2)$$

Dari persamaan diatas dapat dijelaskan bahwa C merepresentasikan *class,* sedangkan F1 sampai dengan Fn merepresentasikan petunjuk yang digunakan untuk klasifikasi. Rumus lain dari teorema *bayes* adalah sebagai berikut :

$$P(Fi|Fj) = \frac{P(Fi \cap Fj)}{P(Fj)} = \frac{P(Fi)P(Fj)}{P(Fj)} = P(Fi) \quad (2.3)$$

Untuk i ≠ j sehingga menjadi sebagai berikut :

$$P(Fi|C, Fj) = P(Fi|C) \tag{2.4}$$

Untuk klasifikasi menggunakan data kontinyu menggunakan rumus *Densitas Gauss* yaitu sebagai berikut :

$$P(Xi|\ Yi) = \frac{1}{\sqrt{2\pi\sigma^2}} exp^{\frac{-(xi-\mu ij)^2}{2\sigma^2 ij}} \tag{2.5}$$

P : merupakan peluang

Xi : merupakan atribut ke i

xi : merupakan nilai atribut ke i

Y : merupakan kelas yang dicari

yi : merupakan sub kelas yang dicari

μ : merupakan rata-rata dari seluruh atribut

σ : merupakan standar deviasi yang digunakan untuk menyatakan variasi pada seluruh atribut

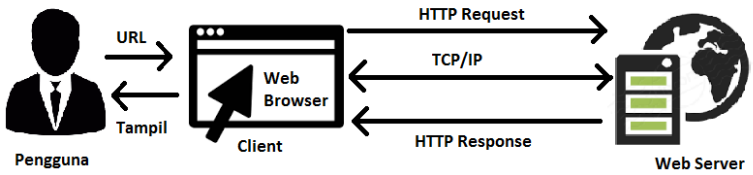
Pada penelitian ini metode *naïve bayes* digunakan untuk melakukan klasifikasi apakah permintaan dari pengguna termasuk permintaan serangan atau permintaan normal. *Naïve bayes* bekerja membutuhkan *data set,* data tersebut diambil dari *log server* yang kemudian data dilakukan ekstraksi fitur. Data yang sudah diekstrak akan dipisah menjadi beberapa *attribute.* Pada penilitian ini terdapat lima *attribute* yang terdiri dari *comment, operator, logical operator, keyword* dan *user agent.*

2.2.3 Teknologi Web

Aplikasi web merupakan aplikasi yang tersimpan pada *web server* dan dieksekusi oleh *web server.* Setiap permintaan pengguna melalui *web browser* akan ditanggapi oleh aplikasi web, kemudian akan dieksekusi oleh *web server.* Permintaan akan dikembalikan kepada pengguna dan ditampilkan pada *web browser* [1]. *Client* dapat melakukan permintaan halaman web melalui *web*

browser. *Web browser* berfungsi untuk menerima informasi dari *web server* kemudian informasi disajikan kepada *client.*

Cara kerja dari suatu web yaitu pengguna terhubung pada suatu jaringan internet. Setelah itu pengguna membuka *browser* dan memasukkan alamat web yang ia tuju. Selanjutnya permintaan akan diteruskan pada TCP/IP yang kemudian data akan dilakukan pengecekan seperti sumber data, tujuan data, dan konten yang ingin dituju. Setelah itu akan diterima oleh *web server,* data TCP/IP kemudian diolah dan disesuaikan menurut permintaan yang dikirimkan. Kemudian data akan dikirimkan kembali melalui TCP/IP kepada pengguna. Permintaan yang diminta akan ditampilkan kepada pengguna melalui *browser*.



Gambar 2.2 Ilustrasi cara kerja web

Aplikasi berbasis web dibangun oleh 2 jenis *scripting* yang terdiri dari :

a) *Server Side Scripting* yang merupakan pemrograman yang dilakukan pada sisi *server*. Pemrograman pada *server* ini hanya dapat dieksekusi oleh *server* dan tidak akan diketahui pada sisi *client.* Bahasa pemrograman yang digunakan pada sisi *server* adalah PHP, ASP, ASP.NET, JSP, Python. Pemrograman pada sisi *server* ini mempunyai tujuan untuk membuat konten aplikasi berbasi web menjadi dinamis sesuai permintaan dari *client.*
b) *Client Side Scripting* yang merupakan pemrograman yang dilakukan pada sisi *client.* Pemrograman pada *client* ini dieksekusi pada *client* ketika membuka *browser* dan melakukan permintaan halaman web. *Client side scripting* membuat tampilan web menjadi lebih interaktif sehingga memudahkan pengguna dalam mengakses dan melakukan permintaan pada web. Bahasa pemrograman yang digunakan untuk melakukan *client side scripting* biasanya *javascript*, *jquery*, css, html.

Arsitektur Web berfokus pada teknologi dan prinsip dasar yang mendukung web tersebut sehingga dapat diakses oleh *client*. Terdapat beberapa bagian dari arsitektur pada web yang terdiri dari:

a) *Hypertext Transfer Protocol* (HTTP)

   HTTP merupakan protokol yang berbasis TCP yang digunakan untuk mengambil file berada di *web server,* selain itu HTTP juga dapat digunakan untuk menampung *headers* dari pengguna untuk menyampaikan informasi pengguna. Selain itu HTTP merupakan protokol yang menempati pada lapisan aplikasi digunakan untuk mengatur komunikasi antara *client* dan *server.* HTTP bertugas mengatur permintaan dari *client* ke *server,* kemudian *server* akan melakukan respon dengan mengirimkan data yang diminta oleh *client* berupa file HTML, file tersebut akan tampil dalam *browser* komputer pengguna.

b) *HyperText Markup Language* (HTML)

   HyperText Markup Language (HTML) merupakan bahasa pemrograman web yang berfungsi untuk membuat suatu konten web agar halaman dapat ditampilkan pada *web browser* [19]. Halaman web yang tampil pada browser berasal dari HTML yang kemudian diterjemahkan oleh *browser* sehingga menampilkan tampilan yang dapat dimengerti oleh pengguna. HTML disimpan pada suatu dokumen dengan format .html, yang didalamnya terdapat banyak *tag* atau simbol tertentu untuk merepresentasikan fungsi tertentu.

c) *Hypertext Preprocessor* (PHP)

   PHP merupakan bahasa pemrograman *server side scripting* yang dieksekusi pada sisi server. Ketika halaman yang mengandung PHP dibuka maka prosesor akan menerjemahkan dan melakukan eksekusi semua perintah yang terdapat pada halaman tersebut, setelah dieksekusi dan diproses oleh *server* maka akan ditampilkan ke *web browser* berupa halaman HTML [20]. PHP berfungsi untuk membuat konten suatu web lebih dinamis dibandingkan dengan web yang hanya menggunakan HTML saja. Maksutnya konten dinamis adalah, halaman mampu berubah-rubah sesuai masukan dari pengguna. Dengan adanya PHP, web dapat menyimpan data ke dalam *database.*

d) *Database*

Database terdiri dari dua kategori yang terdiri dari *database flat* dan *database* relasional. *Database relational* contohnya adalah *mysql, oracle, sql server* dan lainnya. Untuk *database relational* terdapat tabel-tabel yang berfungsi untuk menyimpan data. Pada setiap tabelnya terdapat kolom-kolom dan baris. Kolom digunakan untuk mendefinisikan jenis informasi apa yang akan disimpan. Sedangkan baris adalah nilai masukan yang tersimpan pada tabel tesebut [20]. *Database* berguna untuk meyimpan informasi yang logis. Pada web dinamis, *databse* digunakan untuk penyimpanan data yang dilakukan oleh pengguna. Selain itu data yang tersimpan pada *database* juga dapat ditampilkan dengan menggunakan bantuan bahasa pemrograman PHP, *javascript* sehingga dapat tampil pada *browser client*. Pada era *modern,* rata-rata sudah menggunakan *database* dalam web yang ia kelola sehingga memudahkan pengguna untuk memasukkan data dan menampilkan data secara dinamis.

e) *Web Server*

Web server merupakan penyedia layanan web yang berfungsi untuk mengelola dan melakukan respon terhadap permintaan *web client* [21]. Permintaan yang dikirimkan oleh *client* akan diproses oleh *web server*, kemudian permintaan akan dipenuhi dengan mengirimkan file yang akan ditampilkan secara otomatis oleh *web browser*. Contoh *web server* yang sering digunakan adalah *apache, nginx, IIS, tomcat* dan lainnya.

2.2.4 Serangan Web

Serangan web merupakan suatu aktivitas yang dilakukan oleh *attacker* yang memanfaatkan suatu kelemahan dan kerentanan yang terdapat suatu web. Serangan web beresiko terhadap aspek kerahasian, ketersedian dan integritas. Tujuan serangan web bermacam-macam seperti melakukan pengambilkan akses kontrol pada suatu web, melakukan penyalahgunaan data yang tersimpan pada web, melakukan pencurian, melakukan modifikasi data pada web hingga melakukan perusakan web. Serangan web berfokus pada apalikasi itu sendiri dan fungsinya pada layer 7 *OSI layer*. Terdapat beberapa ancaman serangan pada suatu web yang terdiri sebagai berikut:

a) *Cross Site Scripting* (XSS)

Serangan XSS merupakan serangan dengan cara menyuntikan *script* ke aplikasi web. *Script* yang disuntikan akan tersimpan pada aplikasi web dan akan diproses ketika halaman tersebut diakses oleh pengguna lain melalui *web browser* [22]. Serangan XSS terjadi ketika seorang *attacker* mengirimkan kode berbahaya umumnya dalam bentuk *script* sisi *browser* seperti *javascript* dan *HTML*. Kerentanan ini disebabkan masukkan dari pengguna tidak divalidasi atau encoding, sehingga *attacker* dapat melakukan injeksi *script* yang berbahaya pada masukan tersebut. Jenis kerentanan ini dimanfaatkan oleh *attacker* untuk mengelabui korban (*end user*) untuk mengirimkan kode berbahaya. *Browser* yang dimiliki *end user* tidak memiliki kemampuan untuk mendeteksi bahwa kode tersebut tidak terpercaya, kemudian akan dieksekusi kode tersebut oleh *browser*. Karena *browser* tersebut hanya dapat membaca URL situs yang terpercaya, kode tersebut dapat digunakan untuk mengakses *cookie, session token,* dan informasi sensitif lainnya yang tersimpan pada *browser* dan yang digunakan untuk mengakses situs tersebut. *Attacker* dapat memanfaatkan *cookie* dari korban untuk melakukan login ke situs tersebut. Menurut laporan dari Akamai tahun 2016 serangan *Cross Site Scripting* menempati peringkat ketiga serangan web terpopuler dengan persentase 7.84% pada protokol HTTP dan 7.15% pada protokol HTTPS.

b) *Cross Site Request Forgery* (CSRF)

Serangan CSRF merupakan jenis serangan yang memaksa korban (*end user)* untuk melakukan eksekusi tindakan yang tidak diinginkan pada aplikasi web yang saat ini sedang diotentifikasi. Serangan CSRF secara khusus memaksa untuk melakukan permintaan perubahan kondisi, bukan untuk melakukan pencurian data [23]. Contoh dari serangan ini adalah ketika *attacker* mengirimkan URL untuk memaksa korban melakukan *favorite* pada toko online mereka pada *ecommerce* tertentu. Apabila URL tersebut diakses oleh pengguna biasa atau korban, maka secara otomatis si korban akan melakukan *favorite* kepada toko online *attacker*. Serangan ini perlu dilakukan dengan bantuan *social engineering* untuk mengelabui dan memaksa korban dalam mengakses

suatu URL. Serangan ini terjadi karena tidak adanya CSRF *token* yang *random*. Menurut laporan dari OWASP tahun 2017, CSRF masuk nominasi *top ten* kerentanan yang sering digunakan lebih tepatnya pada posisi 8.

c) *Remote File Inclusion* (RFI)

Serangan RFI merupakan jenis serangan yang digunakan oleh *attacker* untuk menyisipkan file ke suatu *server* melalui *remote* URL [24]. File yang disisipkan mengharuskan *web server* untuk melakukan perintah *server side scripting* seperti PHP melakukan eksekusi file tersebut. Setelah dieksekusi oleh *server side,* maka seorang *attacker* akan mendapatkan akses ke suatu *server* tersebut walaupun hanya mendapatkan akses sebagai *limid user*. Serangan ini disebabkan karena kesalahan dalam pengkodean program di level *server side,* kesalahannya terletak pada pendeklarasian *variable.* Pendefinisian *variable* yang tidak benar pada suatu aplikasi web dapat dimanfaatkan oleh *attacker* untuk melakukan serangan jenis ini. Menurut laporan dari Akamai tahun 2016, RFI menempati peringkat ke enam serangan web yang popular dengan persentase 1.36% pada protokol HTTP dan 1.65% pada protokol HTTPS.

d) *Remote Code Execution* (RCE)

Serangan RCE merupakan serangan yang dimanfaatkan oleh *attacker* untuk melakukan injeksi *code* yang tujuannya dapat dieksekusi oleh suatu sistem operasi untuk mendapatkan suatu akses kedalam *server* [5]. Misalnya *attacker* melakukan injeksi dengan perintah *id,* apabila perintah tersebut dieksekusi maka akan mendapatkan informasi dari *id user* yang berjalan pada tugas tersebut. Serangan ini terjadi karena tidak adanya validasi dalam masukan pengguna. Menurut laporan dari *Trustwave* pada tahun 2016, serangan *remote access* yang terkadang masuk dalam kategori RCE menempati peringkat pertama dengan persentase 13%.

e) SQL Injection

Serangan *SQL Injection* merupakan jenis serangan yang paling sering digunakan untuk melakukan penerobosan ke suatu sistem web secara tidak sah. Serangan *SQL Injection* merupakan jenis serangan injeksi ke suatu aplikasi web dimana *attacker* dapat mengeksekusi pernyatan SQL yang berbahaya [25].

Melalui serangan injeksi ini dapat dimanfaatkan oleh *attacker* untuk melewati otentifikasi dan otorisasi pada suatu aplikasi web. Selain itu, serangan ini dapat dimanfaatkan untuk melakukan pengambilan suatu data yang tersimpan pada *database server*. *SQL injection* dapat digunakan untuk melakukan manipulasi data seperti menambah data, mengubah data dan menghapus data. *SQL injection* merupakan serangan yang dapat berefek pada integritas suatu data.

2.2.5 *SQL Injection*

Serangan *SQL Injection* merupakan jenis serangan yang paling sering digunakan untuk melakukan penerobosan ke suatu sistem web secara tidak sah. Serangan *SQL Injection* merupakan jenis serangan injeksi ke suatu aplikasi web dimana *attacker* dapat mengeksekusi pernyatan SQL yang berbahaya [25]. Melalui serangan injeksi ini dapat dimanfaatkan oleh *attacker* untuk melewati otentifikasi dan otorisasi pada suatu aplikasi web. Selain itu, serangan ini dapat dimanfaatkan untuk melakukan pengambilan suatu data yang tersimpan pada *database server*. *SQL injection* dapat digunakan untuk melakukan manipulasi data seperti menambah data, mengubah data dan menghapus data. *SQL injection* merupakan serangan yang dapat berefek pada integritas suatu data. Karena melalui *SQL injection* mendapatkan akses secara tidak sah ke sensitif data seperti data rahasia, data pelanggan dan sensitif data lainnya. Serangan ini terjadi karena terdapat kesalahan pada suatu pengkodean aplikasi web sehingga masukan dari pengguna tidak dilakukan validasi dan penyaringan, sehingga masukan pengguna langsung dieksekusi oleh aplikasi web dan *database* hingga menyebabkan terjadinya injeksi tersebut.

Cara kerja dari serangan *SQL injection* sendiri yaitu *attacker* mencari masukan pada aplikasi web yang didalamnya mengandung *query* SQL, misalnya menu pencarian, *login*, dan masukan lainnya. Agar serangan ini terjadi seorang *attacker* memasukkan *payload* yang akan dimasukkan sebagai bagian dari *query SQL* pada masukan aplikasi web tersebut. Setelah itu *query* asli dan *payload* injeksi akan dieksekusi dan diproses oleh *database server*. Hasil proses dari *database server* akan ditampilkan pada web aplikasi. Apabila injeksi tersebut berhasil, maka

akan memunculkan *output* sesuai *payload* yang diinjeksikan tadi, misalnya nama *database* yang digunakan. *SQL Injection* mempunyai 6 tipe yang terdiri dari: [26]

a) *Tautology-based SQL Injection*

Tautology merupakan formula yang benar dalam interpretasi yang mungkin dapat terjadi. Tujuan dari serangan tipe *tautology-based* adalah untuk melakukan mengidentifikasi parameter injeksi, melakukan ekstrak data dan melakukan *bypass authentication* untuk mendapatkan akses secara tidak sah. Dalam serangan ini, kode disuntikan menggunakan operator *OR* bersyarat sehingga *query* akan mengembalikan nilai *true.* Serangan dengan jenis ini biasanya sering digunakan untuk melakukan *bypass authentication* pada klausa *WHERE* dari *query SQL.* Permintaan merubah kondisi semula menjadi *tautology* yang menyebabkan baris dalam tabel pada *database* terbuka untuk pengguna yang tidak sah. Serangan tipe ini memiliki bentuk "*OR* <perbandingan ekspresi>", perbandingan satu atau lebih operator relasional untuk membandingkan operan dan menghasilkan kondisi yang selalu benar. Berikut ini contoh kasus *SQL injection attack* tipe *tautology-based attack* untuk mendapatkan akses secara tidak sah pada aplikasi web yang mengakibatkan *attacker* melewati otentifikasi pada aplikasi web.

Tabel 2.2 *Pseudecode Vulnerability SQL Injection*

Pseudecode
username = request.POST['username'] *password = request.POST['password']* *# Query SQL yang rentan terhadap serangan SQL Injection* *sql = "SELECT * FROM table_users WHERE username= 'username' AND password= 'password'"* *database.execute(sql)*

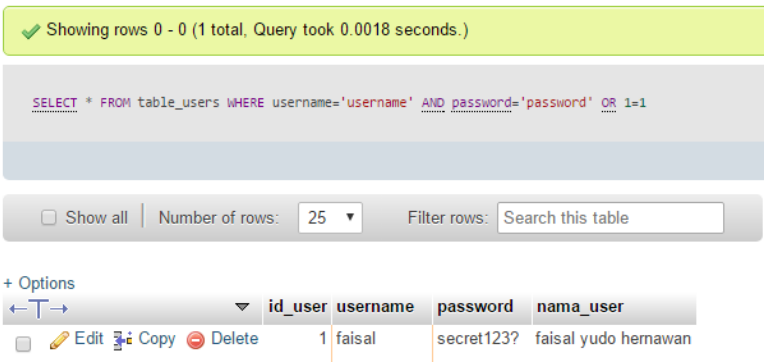
Pseudecode diatas bertujuan untuk melakukan *login* pada suatu aplikasi web karena terdapat 2 parameter yang digunakan yaitu *username* dan *password.* Pada pernyataan SQL diatas apabila dieksekusi akan menampilkan data dari *table users* apabila *username* dan *password* yang dimasukkan pengguna cocok

dengan *username* dan *password* yang tersimpan pada *database.* Apabila *username* dan *password* salah maka pengguna tidak dapat melakukan *login* ke aplikasi web. Namun, kode diatas memiliki kerentanan yang mengakibatkan terjadinya serangan *SQL injection.* Parameter *username* dan *password* tidak dilakukan validasi sehingga pengguna dapat melakukan masukan berbagai karakter, dengan begitu pengguna dapat melakukan serangan *SQL injection. Attacker* dapat mengirimkan *payload* yang berbahaya yang akan merubah pernyataan SQL yang nantinya akan dieksekusi oleh *database server.* Contoh dari *payload SQL injection* untuk menetapkan kolom *password* untuk *password* yaitu *password' OR 1 = 1.* Hal itu akan mengakibatkan *query SQL* berikut yang akan dijalankan pada *database server.*

Tabel 2.3 Eksekusi *Payload SQL Injection*

Eksekusi *Payload SQL Injection*
*SELECT * FROM table_users WHERE username='username' AND password='**password' OR 1=1'***

Setelah *query* diatas dieksekusi oleh *database server,* hasilnya akan dikembalikan ke aplikasi web yang mengakibatkan otentifikasi terlewati. Hal ini disebabkan *payload 'OR 1 = 1* akan mengambalikan nilai *true* sehingga proses otorisasi terlewati. Berikut adalah hasil pemrosesen *query* yang dilakukan oleh *database mysql* yang ditunjukan pada Gambar 2.3.



Gambar 2.3 *Tautology-based SQL Injection*

b) *Piggy-backed Queries / Statement Injection*

Jenis serangan *sql injection* tipe ini adalah menyuntikan *query* tambahan setelah *query* asli. Serangan ini bertujuan untuk melakukan ekstrak data, melakukan modifikasi data, melakukan eksekusi perintah secara jarak jauh, hingga melakukan serangan *denial of service* yang menyebabkan lumpuhnya suatu sistem. Alur dari serangan ini adalah, sistem membaca *query* asli dan melakukan validasi kemudian dijalankan secara normal, kemudian sistem akan melakukan pembacaan *query* selanjutnya yang disuntikan oleh peretas dan sistem akan melakukan eksekusi *query* tersebut setelah *query* pertama dijalankan. Jenis serangan ini sangat berbahaya, jika peretas menemukan dan mampu melakukan serangan maka peretas dapat melakukan suntikan *query SQL* yang menyebabkan sistem lumpuh atau bahkan menghapus *database*. Berikut ini adalah contoh kasus dari serangan *sql injection* tipe *piggy-backed queries.* Terdapat *query sql* asli yang digunakan untuk melakukan *authentication login.*

Tabel 2.4 *Query SQL*

Query SQL asli
*SELECT * FROM table_users WHERE username = 'faisal' AND password = 'secret123?'*

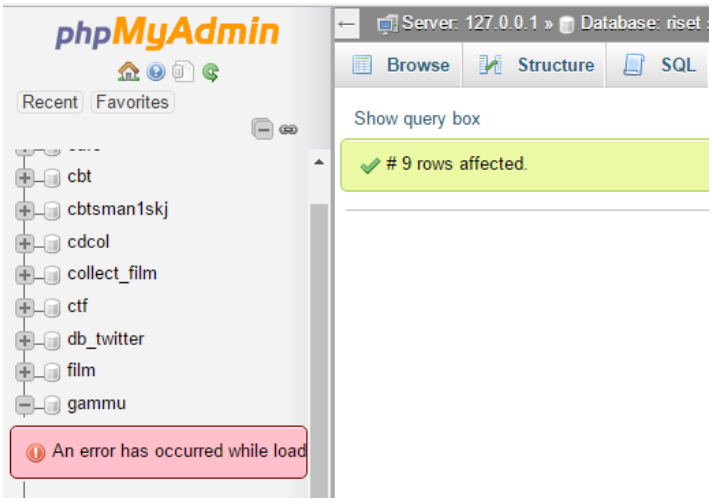
Peretas dapat menyuntikan *query* tambahan dengan cara menambahkan tanda baca titik koma “ ; “ setelah *query* pertama. Pada kasus ini akan melakukan suntikan dengan cara melakukan suntikan pada kolom *password*, sehingga suntikan yang dilakukan oleh peratas adalah memasukkan *password* secret123?;*query* injeksi.

Tabel 2.5 Serangan *piggy-based queries*

Serangan *piggy-based queries*	
Query injeksi	*DROP DATABASE gammu*
Nilai yang dikirimkan ke *database server*	*password; DROP DATABASE gammu*

Serangan *piggy-based queries*	
Query SQL	*SELECT * FROM table_users WHERE username = 'faisal' AND password = 'secret123?'; DROP DATABASE gammu*
Aksi yang dilakukan oleh *database server*	Menampilkan pengguna yang mempunyai *username* dan *password* yang di masukan oleh peretas. Setelah itu menghapus *database* yang bernama gammu.

Berikut adalah hasil pemrosesen *query* yang dilakukan oleh *database mysql* yang ditunjukan pada Gambar 2.4.



Gambar 2.4 *Piggy-backed Queries / Statement Injection*

c) *Union Query*

Serangan *sql injection* tipe *union* merupakan serangan yang melakukan ekploitasi pada parameter yang rentan untuk mengembalikan kumpulan data yang merupakan penyusutan hasil *query* pertama yang merupakan *query* asli dan hasil *query* kedua yang merupakan *query* suntikan. Tujuan dari serangan tipe *union* untuk melakukan ekstrak data dan melakukan *bypass authentication.* Operator *union SQL* menggabungkan hasil dua *query* lebih dan menghasilkan

kumpulan hasil yang mencakup baris yang diambil dari *query* yang berpartisipasi di *union*. Aturan dasar untuk menggabungkan dua atau lebih *query* menggunakan *union*:

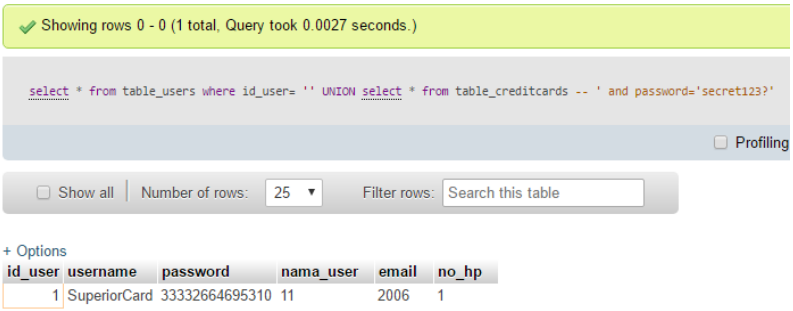
1. Sejumlah kolom dan urutan kolom dari semua *query* harus sama.
2. Tipe data kolom yang melibatkan tabel dalam setiap *query* harus sama atau kompatibel.
3. Biasanya nama kolom yang dikembalikan diambil dari *query* pertama.

Berikut ini adalah contoh dari serangan *sql injection* tipe *union*, penyerang bisa menyuntikkan teks *' UNION SELECT * FROM table_creditcards -- '* kedalam *field id_user* sehingga menghasilkan *query "SELECT * FROM table_users WHERE id_user= '' UNION SELECT * FROM table_creditcards -- ' AND password='secret123?'"* Dengan asumsi bahwa tidak ada id_user yang sama dengan "", *query* pertama yang asli mengembalikan himpunan nihil, sedangkan *query* kedua mengembalikan data dari tabel "table_creditcards". Dalam kasus ini, *database* akan mengembalikan kolom yang terdapat pada tabel *table_creditcars* untuk pengguna yang memupunyai *password secret123?*. *Database* mengambil hasil dari dua *query* ini, menggabungkannya, dan mengembalikannya ke aplikasi.

Tabel 2.6 Eksekusi *Payload SQL Injection*

Eksekusi *Payload SQL Injection*
*SELECT * FROM table_users WHERE id_user= '' **UNION SELECT * FROM table_creditcards** -- ' AND password='secret123?'*

Berikut adalah hasil pemrosesen *query* yang dilakukan oleh *database mysql* yang ditunjukan pada Gambar 2.5.



Gambar 2.5 *Union Query*

d) *Illegal/Logically Incorrect Queries*

Jenis serangan ini biasanya dilakukan oleh peretas untuk mengumpulkan informasi penting tentang jenis dan struktur dari *database* pada suatu aplikasi web. Serangan ini bertujuan untuk melakukan identifikasi parameter injeksi, melakukan pemindaian *database* dan melakukan ekstrak data. Kerentanan yang mampu dieksploitasi oleh serangan ini adalah halaman kesalahan *default* yang dikembalikan oleh *server*. Pesan kesalahan dihasilkan seringkali dapat mengungkapkan parameter yang rentan atau dapat disuntikkan oleh peretas. Informasi kesalahan tambahan, yang awalnya ditujukan untuk membantu programmer melakukan *debugging* aplikasi dapat dimanfaatkan oleh peretas untuk mendapatkan informasi tentang skema *back-end database*. Saat melakukan serangan ini, penyerang mencoba menyuntikkan pernyataan yang menyebabkan sintaks, jenis konversi, atau kesalahan logis ke dalam *database*. Kesalahan sintaks dapat digunakan untuk mengidentifikasi parameter injeksi. Jenis kesalahan dapat digunakan untuk menyimpulkan jenis data kolom tertentu atau untuk mengekstrak data. Kesalahan logis sering mengungkapkan nama tabel dan kolom yang menyebabkan kesalahan. Contoh serangan jenis ini adalah menggunakan *CONVERT*, seperti melakukan injeksi "*convert(int,(select top 1 name from sysobjects where xtype='u'))*" pada kolom *id_user*. *Query* yang dihasilkan adalah sebagai berikut *"SELECT username FROM table_users WHERE username='' AND password='' AND id_user= CONVERT (INT,(SELECT TOP 1 name FROM sysobjects WHERE xtype='u'))"*. Dalam serangan, *query* yang disuntikkan mencoba untuk mengekstrak tabel pengguna yang memiliki huruf pertama u dari tabel *metadata database*. *Query* kemudian

mencoba untuk mengubah nama tabel ini menjadi bilangan bulat. Karena ini bukan tindakan yang legal maka jenis konversi, *database* akan mengeluarkan pesan *error.*

Tabel 2.7 Eksekusi Payload *SQL Injection*

Eksekusi *Payload SQL Injection*
SELECT nama FROM table_users WHERE username=’’ AND password=’’ AND id_user= ***CONVERT (INT,(SELECT TOP 1 name FROM sysobjects WHERE xtype=’u’))****”*

e) *Inference*

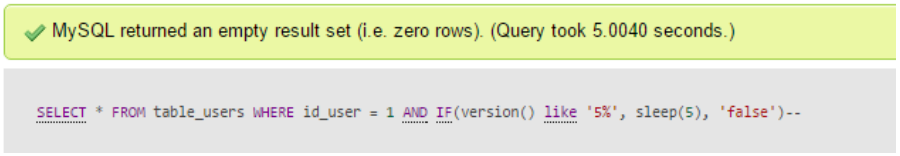
Pada serangan *sql injection* jenis ini, serangan tersebut diterapkan pada *database* yang aman yang tidak mengembalikan umpan balik mengeluarkan pesan kesalahan deskriptif. Serangan ini bertujuan untuk melakukan ekstrak data, identifikasi skema dan identifikasi parameter yang dapat diinjeksi. Serangan biasanya dibuat dengan gaya pernyataan palsu yang sebenarnya. Setelah menemukan parameter yang rentan, penyerang dapat menyuntikkan berbagai kondisi melalui *query* dan mengamati situasinya dengan saksama. Jika pernyataan dievaluasi ke *true*, halaman tetap berfungsi normal. Jika salah, halaman berperilaku berbeda secara signifikan dari fungsi normal. Jenis injeksi ini disebut *Blind SQL Injection*. Ada jenis serangan inferensi lain yang disebut *Time Attack*. Dalam metode ini, penyerang merancang pernyataan kondisional dan menyuntikkan melalui parameter yang rentan dan mengumpulkan informasi berdasarkan penundaan waktu dalam respon *database*. Berikut contoh serangan yang menggunakan *time attack.*

Tabel 2.8 Eksekusi *Payload SQL Injection*

Eksekusi *Payload SQL Injection*
*SELECT * FROM table_users WHERE id_user = 1* ***AND IF(version() like '5%', sleep(5), 'false')--***

Query diatas digunakan oleh peretas untuk melakukan pengecekan apakah sistem tersebut menggunakan *database mysql* versi 5.x atau tidak, apabila

menggunakan versi tersebut maka akan menunda eksekusi dalam waktu 5 detik seperti pada Gambar 2.6.



Gambar 2.6 *Inference*

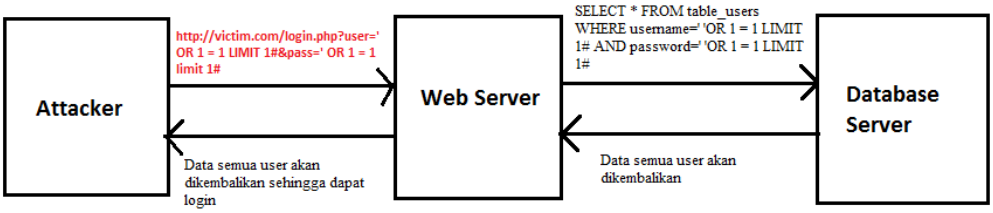
f) *Stored Procedure Injection*

Serangan *sql injection* jenis ini menjalankan prosedur yang tersimpan pada *database*. Serangan ini bertujuan untuk mendapatkan hak istimewa yang lebih tinggi, melakukan *denial of service* yang menyebabkan sistem *database* lumpuh, serta melakukan eksekusi *command* sistem secara jarak jauh. *Stored procedure* merupakan subrutin yang tersedia untuk aplikasi yang digunakan untuk mengakses sistem *database* relasional. Penggunaan tipikal untuk prosedur tersimpan mencakup validasi data atau mekanisme kontrol akses. Selanjutnya, prosedur yang tersimpan dapat mengkonsolidasikan dan memusatkan logika yang pada awalnya diimplementasikan dalam aplikasi. Proses ekstensif atau kompleks yang memerlukan eksekusi beberapa pernyataan SQL dipindahkan ke prosedur tersimpan, dan semua aplikasi memanggil prosedur. Jenis prosedur tersimpan injeksi SQL mencoba untuk mengeksekusi prosedur tersimpan yang ada di *database*. Sebagian besar *database* memiliki seperangkat prosedur standar yang memperluas fungsionalitas *database* dan memungkinkan terjadinya interaksi dengan sistem operasi. Penyerang awalnya mencoba untuk menemukan tipe *database* dengan metode injeksi lain. Setelah penyerang menentukan *database* mana yang digunakan di backend maka ia mencoba menjalankan berbagai prosedur melalui kode yang disuntikkan. Karena prosedur yang tersimpan ditulis oleh pengembang, oleh karena itu prosedur ini tidak membuat *database* rentan terhadap serangan injeksi SQL. Contoh serangan yang dapat dilakukan adalah sebagai berikut.

Tabel 2.9 Eksekusi *Payload SQL Injection*

Eksekusi *Payload SQL Injection*
*SELECT * FROM table_users WHERE username = '' and password =* ***''; SHUTDOWN; -- '***

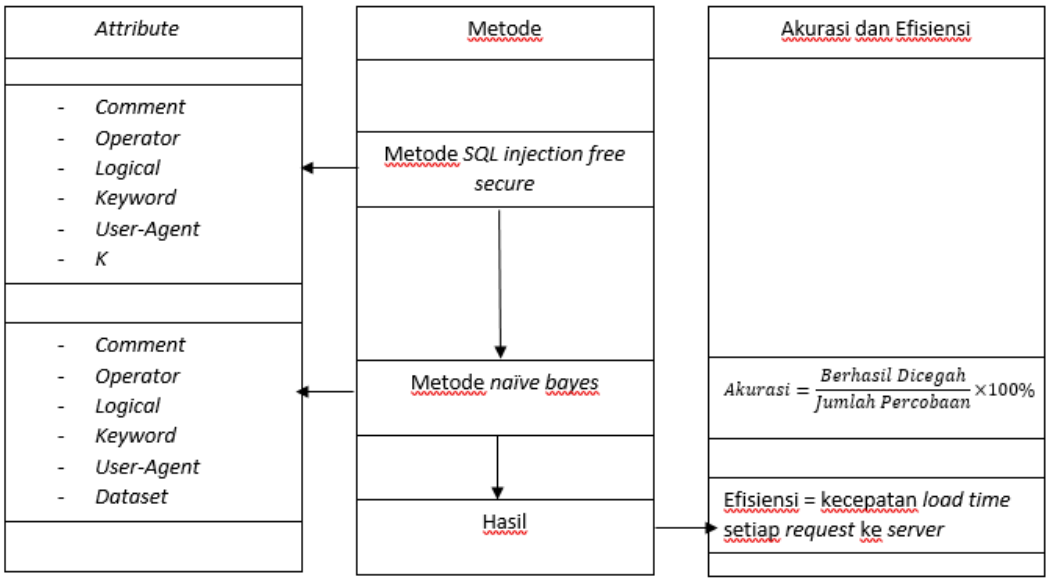
Mekanisme serangan *sql injection* menggunakan banyak mekanisme yang berbeda. Injeksi melalui *input* pengguna: penyerang menyuntikkan perintah SQL dengan menyediakan *input* pengguna yang sesuai. Aplikasi web dapat membaca masukan pengguna dengan beberapa cara berdasarkan lingkungan tempat aplikasi diterapkan. Pada sebagiaan besar yang menargetkan aplikasi web, masukan pengguna biasanya berasal dari pengiriman *form* yang dikirim ke aplikasi web melalui permintaan HTTP *GET* atau *POST*. Aplikasi web umumnya dapat mengakses masukan pengguna yang terdapat dalam permintaan ini karena mereka akan mengakses variabel lain di lingkungan. Injeksi melalui variabel *server*: Variabel *server* merupakan kumpulan variabel yang mengandung HTTP, *header* jaringan, dan variabel lingkungan. Aplikasi web menggunakan variabel *server* ini dengan berbagai cara, seperti statistik penggunaan *logging* dan mengidentifikasi tren penjelajahan. Jika variabel-variabel ini dicatat ke *database* tanpa sanitasi, ini bisa membuat kerentanan injeksi SQL. Karena penyerang dapat menempa nilai yang ditempatkan di header HTTP dan jaringan, mereka dapat memanfaatkan kerentanan ini dengan menempatkan SQLIA langsung ke *header*. Saat *query* untuk login server. Variabel dikeluarkan ke *database*; Serangan di *header* palsu kemudian dipicu. Injeksi melalui *cookies*: *Cookies* berisi informasi keadaan yang dihasilkan oleh aplikasi web dan disimpan pada mesin klien. Ketika klien kembali ke aplikasi web, *cookies* dapat digunakan untuk mengembalikan informasi negara klien. Karena klien memiliki kendali atas penyimpanan *cookie*, klien jahat dapat merusak isi *cookies*. Jika aplikasi web menggunakan isi *cookie* untuk membuat *query* SQL, penyerang dapat dengan mudah mengirimkan serangan dengan menanamkannya di cookie



Gambar 2.7 Proses *SQL Injection Attack*

2.3 Kerangka Pemikiran

Berikut ini adalah kerangka pemikiran penelitian yang akan digunakan untuk penelitian, sesuai Gambar 2.8.



Gambar 2.8 Gambar Kerangka Pemikiran