

Menangkal Serangan *SQL Injection* dengan *Parameterized Query*

Yulianingsih^{#1}

[#]Teknik Informatika, Fakultas Teknik dan MIPA Universitas Indraprasta PGRI
Jalan Nangka no. 58 Tanjung Barat Jagakarsa Jakarta Selatan

¹yuliagniya@yahoo.co.id

Abstrak— Semakin meningkat pertumbuhan layanan informasi maka semakin tinggi pula tingkat kerentanan keamanan dari suatu sumber informasi. Melalui tulisan ini disajikan penelitian yang dilakukan secara eksperimen yang membahas tentang kejahatan penyerangan database secara *SQL Injection*. Penyerangan dilakukan melalui halaman autentikasi dikarenakan halaman ini merupakan pintu pertama akses yang seharusnya memiliki pertahanan yang cukup. Kemudian dilakukan eksperimen terhadap metode *Parameterized Query* untuk mendapatkan solusi terhadap permasalahan tersebut.

Kata kunci— Layanan Informasi, Serangan, eksperimen, *SQL Injection*, *Parameterized Query*.

I. PENDAHULUAN

Aplikasi berbasis web berkembang pesat mengikuti pertumbuhan kebutuhan dunia bisnis yang saat ini cenderung mengedepankan layanan berbasis *online* untuk menjawab kebutuhan *customer* mendapatkan layanan yang cepat, tepat dan aman. Disisi lain peningkatan tersebut mengakibatkan meningkatnya ide kreatif dari para penyerang (*attacker*) yang menyalah gunakan kesempatan tersebut untuk mendapatkan segala sesuatu dengan cara ilegal. Dampak dari kegiatan tersebut tentu saja menimbulkan kerugian pada kedua *stackholder* yaitu *produsen* dan *customer*. Munculnya kerentanan tersebut dipengaruhi beberapa faktor antara lain kurangnya pengetahuan *customer* terhadap jenis layanan berbasis web dan kurang siapnya penyelenggara layanan terhadap isu-isu keamanan yang berhubungan dengan bisnis berbasis *online* atau *e-commerce*.

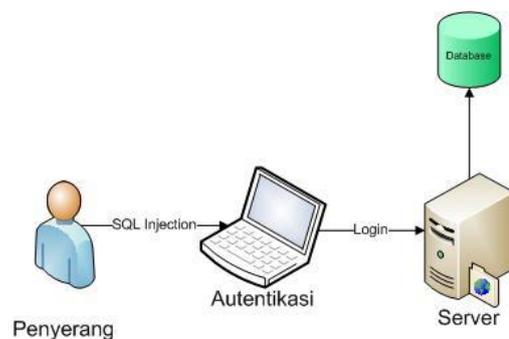
Dengan demikian diperlukan sebuah metode yang mampu memberikan solusi tepat. Untuk dapat menentukan metode yang tepat maka dianggap perlu mengenal karakter dan cara kerja dari kegiatan penyerangan tersebut. Dalam penelitian ini diperkenalkan jenis penyerangan dengan cara *sql injection* secara server side scripting untuk kemudian diberikan solusi menghindarinya. Dalam penelitian ini dianggap perlu mengangkat jenis serangan *SQL Injection* dikarenakan bahwa total serangan terhadap situs-situs yang ada di Indonesia adalah 28.430.843 dan jenis serangan paling besar adalah melalui *SQL* [1].

II. LANDASAN TEORI

Penelitian ini didukung dengan teori-teori sebagai acuan untuk memberikan solusi yang tepat antara lain:

A. *SQL Injection*

Merupakan sebuah kerentanan yang menyebabkan seorang penyerang memiliki kemampuan untuk mempengaruhi query *SQL* yang dikirimkan melalui aplikasi ke database. Dengan kemampuan tersebut seorang penyerang dapat mempengaruhi *syntax SQL*, kekuatan, fleksibilitas dari database pendukung fungsional dan mempengaruhi fungsi sistem operasi yang dialokasikan untuk database. *SQL Injection* tidak hanya mempengaruhi aplikasi web tapi juga semua program lain yang menggunakan kalimat *SQL*. Semua program yang menggunakan input dinamis dari luar (*untrusted*) dapat terserang oleh *SQL*. [2]



Gambar. 1 *SQL Injection* melalui Autentikasi

Gambar 1. Menunjukkan cara kerja seorang penyerang menggunakan *SQL Injection* melalui pintu pertama sebuah situs yaitu halaman login. Melalui autentikasi *sql injection* membentuk logika tertentu untuk dapat masuk kedalam system database.

B. Bagaimana *SQL Injection* dapat mengeksploitasi kerentanan?

Beberapa hal yang membuka kesempatan bagi penyerang untuk melakukan *SQL Injection* antara lain: [3]

- Memanfaatkan keuntungan dari sebuah aplikasi yang tidak terlindungi pada fungsi autentikasi pengguna karena tidak adanya validasi.
- Umumnya seorang penyerang membajak *login field* yang tidak terlindungi untuk memperoleh akses database.
- *SQL Interpreter* tidak dapat membedakan antara perintah yang dimaksud dengan kode yang di-*inject* oleh penyerang

yang kemudian dieksekusi dan mengakibatkan tereksposnya database. Peyerang kemudian dapat mengakali aplikasi untuk mengekstrak semua data, menanamkan *malicious script*.

C. Bahaya SQL Injection

Berikut diberikan beberapa data yang dapat dijadikan acuan tentang bahaya yang pernah terjadi dari tindakan tersebut : [3]

- Ancaman terbesar terhadap aplikasi web pada tahun 2011, menurut Trustwave terjadi 71 serangan SQL Injection setiap jamnya, menurut Imperva
- 97% kebocoran data disebabkan oleh SQL Injection, menurut Nationa Fraud Authority, UK.
- 83% kegiatan hacking situs web yang sukses tahun 2005-2011 menggunakan metode SQL Injection, menurut privacy rights.org
- Serangan-serangan SQL Injection dengan kerugian terbesar antara lain: Card Systems Solutions (2005, 40 juta kartu data kredit berhasil dicuri), TJX (2006, 94 juta kartu), Heartland Payment Systems (2008, 134 juta kartu), RockYou (2009, 32 juta pengguna), Sony (2011, 77 juta pengguna).
- Serangan terautomatisasi berhasil menginfeksi 100.000 web: In 2008, SQL attack menjadi terautomatisasi via botnets. Infeksiasi yang pernah terjadi adalah yang melibatkan lebih dari 500 juta pengguna, dilaporkan 500.000 insiden yang dilaporkan tahun 2008.

Bahkan Seorang penyerang dapat memanfaatkan pesan error yang dikirimkan server ketika dia memberikan query-query illegal untuk mendapatkan nama dan tipe dari kolom pada tabel suatu database [4].

D. Metode Untuk Menghindari SQL Injection

Metode untuk menghindari SQL Injection dapat dilakukan kedalam dua cara yaitu secara *client-side* dan *server-side*. Pada metode client-Side yaitu menerima '*Shadow SQL Query*' dari *server-side* dan melakukan pengecekan terhadap deviasi yang terjadi antara *shadow query* dengan query dinamis yang dibentuk oleh masukan dari pengguna. Jika ditemukan adanya deviasi maka dapat dipastikan bahwa masukannya tidak benar (*Malicious*)[5]. Sedangkan dalam penelitian ini dikedepankan metode secara *server-side*. yaitu melalui sejumlah langkah antara lain:

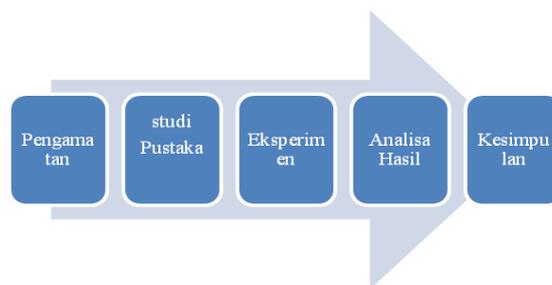
1. *Gunakan Prepared Statement* atau *Parameterized Query*. Lebih sederhana dan lebih mudah untuk dimengerti dibanding query dinamis, *Parameterized query* mengirimkan setiap parameter kedalam lapisan query setelah semua kode SQL telah didefinisikan. Database dapat membedakan antara kode dan data, apapun masukan dari pengguna. Seorang penyerang tidak dapat merubah

maksud query, meskipun *SQL command* telah disusupkan di dalamnya.

2. Lakukan validasi masukan. Autentikasi masukan dari pengguna dengan aturan-aturan yang telah didefinisikan seperti panjang, tipe dan filter lain jika diperlukan.
3. Matikan atau sembunyikan pesan-pesan error yang keluar dari server database.
4. Mengunci atau membatasi database anda. Jika aplikasi web Anda tidak memerlukan akses ke tabel tertentu, maka pastikan bahwa mereka semua tidak memiliki izin untuk itu. Jika hanya read-only maka akan menghasilkan laporan dari tabel hutang account anda maka pastikan anda menonaktifkan insert / update / menghapus akses..

III. METODELOGI PENELITIAN

Sejumlah eksperimen dilakukan dalam penelitian ini untuk mencari pengaruh perlakuan tertentu terhadap yang lain dalam kondisi yang terkendali [6]. Penelitian dilakukan melalui sejumlah langkah yang dijelaskan melalui gambar 2.



Gambar. 2 Langkah Penelitian

Gambar 2. Menunjukkan urutan penelitian yang diawali dengan melakukan pengamatan untuk mengidentifikasi masalah yang ada. Kemudian dilakukan kajian pustaka terkait dengan masalah yang ada untuk menentukan metode terbaik yang dapat dijadikan solusi. Pada tahapan eksperimen solusi tersebut diuji cobakan dengan menyusun rangkaian bahasa pemrograman menggunakan PHP. Hasil eksperimen dianalisa untuk ditarik kesimpulan apakah pengujian yang dilakukan memperoleh hasil sesuai dengan tujuan yang hendak dicapai.

Tahapan eksperimen pada penelitian:

Tahap1.

Memberikan serangan mengikuti pola kejahatan *SQL Injection* yaitu berusaha masuk melalui halaman autentikasi secara illegal.

Tahap 2.

Membangun rangkaian pemrograman yang berisikan metode *parameterized query* dengan php

Tahap 3.

Memberikan serangan mengikuti pola kejahatan *SQL Injection* melalui halaman autentikasi. Dengan cara berusaha memasuki layanan menggunakan password tertentu yang tidak legal dengan kondisi halaman autentikasi sudah ditanamkan *parameterized query*.

Tahap 4.

Mengambil kesimpulan dari keseluruhan pengujian

IV. PEMBAHASAN

Halaman autentikasi merupakan pintu pertama pengamanan yang dimiliki oleh sebuah situs, umumnya yang harus dilakukan oleh pengguna layanan adalah mengisi user dan password. Kedua bagian ini mempunyai prinsip kerja dimana harus memiliki nilai keduanya benar maka benar.

Hasil tahapan eksperimen berikut memberikan pembuktian tersebut.

Tahap 1. Login dengan *SQL Injection*

Mengetikkan kalimat query yang berisi *SQL Injection* kedalam *field password* dengan tujuan melakukan login secara ilegal



Gambar. 3 Hasil Query SQL Injection

Gambar 3. Menunjukkan query yang diberikan kedalam *field password* dengan menggunakan *SQL Injection*. Password dimasukan sejumlah karakter yang menghasilkan logika selalu benar yaitu berupa karakter "password' OR '1'='1" yang bukan merupakan nilai password yang sesungguhnya dan hasil yang diperoleh adalah berhasil login.

Hal ini dikarenakan *SQL Interpreter* tidak dapat membedakan antara perintah yang dimaksud dengan kode yang di-inject oleh penyerang yang kemudian dieksekusi dan mengakibatkan teresposnya database.

Tahap 2. Membangun logika parameterized query

```

1. <?php
2. if(isset($_POST['submit'])){
3. $dbHost = "localhost";
4. $dbUser = "root"; //Database User Name
5. $dbPass = "90super"; //Database Password
6. $dbDatabase = "sqli"; //Database Name
7. //Connect to the atabase
8. $db = new
9. PDO("mysql:dbname=$dbDatabase;host=$dbHost;port=3. 306", $dbUser, $dbPass);

```

```

10. $sql = $db->prepare("SELECT * FROM user
    WHERE username = ? AND password = ? LIMIT
    1");
11. $sql->bindValue(1, $_POST['username']);
12. $sql->bindValue(2, $_POST['password']);
13. $sql->execute();
14. if($sql->rowCount() == 1){
15. $row = $sql->fetch($sql);
16. session_start();
17. header("Location: users_page.php");
18. }else{
19. echo "<br>salah euy";
20. //header("Location: login_page.php");
21. exit;
22. }
23. }else{
24. header("Location: index.php");
25. exit;
26. }

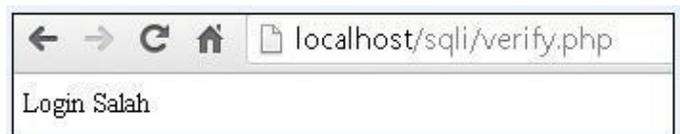
```

Baris ke-10 pada listing program merupakan langkah memberikan isi field password kepada suatu parameter tertentu.

Baris ke-11 pada *listing program* dilakukan proses pembersihan query yang masuk dengan memilah sejumlah karakter yang tidak berhak untuk berada pada posisi tersebut. Bilamana ditemukan maka hasil query dianggap tidak benar.

Tahap 3. Memberikan serangan kedua dengan cara yang sama dengan langkah pertama setelah adanya *parameterized query* pada listing program.

Mengetikkan kembali kalimat query yang berisi *SQL Injection* kedalam *field password* yang sudah dilakukan pemberian *parameterized query* dengan tujuan login secara ilegal. Dan mendapatkan hasil login salah.



Gambar. 4 Hasil Query SQL Injection dengan Parameterized Query

Gambar 4. Menunjukkan hasil dari *query injection* terhadap halaman autentikasi yang telah diberikan *parameterized query*.

Pada tahapan ini membuktikan bahwa serangan melalui password dapat dicegah dengan baik.

Tahap 4. Menganalisa data hasil eksperimen yang diperoleh kemudian menarik kesimpulan.

Ditemukan pembuktian bahwa *parameterized query* yang digunakan berhasil melakukan fungsinya untuk melakukan pertahanan dari usaha penyerangan secara *SQL Injection*.

III. KESIMPULAN

Berdasarkan hasil eksperimen didapat sejumlah kesimpulan antara lain :

1. Halaman autentikasi merupakan pintu pertama keamanan suatu data harus memiliki fasilitas pertahanan yang tinggi yang mampu mengikuti perkembangan teknologi dengan benar yang bertujuan mengikuti pola berpikir para penyerang yang semakin banyak karakteristik dan cara kerjanya nya.
2. Menganggap suatu metode atau jenis penyerangan yang telah usang atau sederhana merupakan suatu tindakan ceroboh yang dapat merugikan diri sendiri.
3. Menentukan konsep pertahanan pada lapisan awal suatu layanan berbasis web perlu memperhatikan beberapa hal anantara lain : kenyamanan pengguna layanan dan pengaruhnya terhadap kecepatan proses.

REFERENSI

- [1] *Indonesia Cyber Security Report 2015*, Id-SIRTII/CC, 2015
- [2] Justin Clarke, (2009), *SQL Injection Attacks and Defense*, Syngress Publisher.
- [3] *SQL Injection Fact Sheet*, Veracode, 2012
- [4] Yudiantoro, Tri Raharjo, *SQL Injection pada sistem keamanan database*. Jurnal Teknologi Informasi dan Komunikasi, LPPM STMIK ProVisi: Semarang, 2013, vol 4 No.2.
- [5] Hossain Shariar, Sarah North, Wei-Chuen Chen, *Early Detection of SQL Injection Attacks*, International Journal of Network & Its Application (IJNSA), 2013, vol 5 no 4.
- [6] Sugiyono. *Metode Penelitian Kuantitatif Kualitatif dan R&D*. Bandung : Alfabeta. 2009.