ISSN : 2085-4218

Analisis Penanganan SQL Injection pada Basis Data MySQL dengan Framework Code Igniter dan PHP

Muhammad Rizal Efendi[1,*], Leanna Vidya Yovita[1], Hafidudin[2]
1 Fakultas Teknik Elektro, UniversitasTelkom. JL. Telekomunikasi No.1, Bandung 40257
2 Fakultas Ilmu Terapan, UniversitasTelkom. JL. Telekomunikasi No.1, Bandung 40257
* E-mail : rizalafandi01@gmail.com

Abstrak. *SQL Injection* adalah sebuah metode menyisipkan *query sql* kedalam sebuah aplikasi web melalui *form* input dan *url*(*uniform resource locator*) untuk medapatkan informasi pada basis data. Pengamanan aplikasi web dapat dilakukan dengan memasang *firewall, antivirus,* atau *software* sejenis pada *server*. Namun yang terpenting adalah membuat kode program terbebas dari kesalahan. Karena *firewall* terbaik tidak akan berguna jika kode program tidak terbebas dari kesalahan. PHP dan Code Igniter adalah contoh bahasa pemograman yang sering digunakan oleh *developer* dalam membuat suatu aplikasi web, itu semua dipengaruhi beberapa faktor diantaranya mudah digunakan, ukuran relatif kecil dan cepat, kemudahan instalasi, terdapat *library* validasi, dokumentasi lengkap, mendukung PHP 4 dan PHP 5, dan menyediakan fungsi enkripsi, *session, cookies, xss filtering*. Pada penelitian ini dilakukan analisis teknik penanganan *SQL Injection* pada basis data *MySQL* dengan *framework Codeigniter* dan *PHP* yang mana digunakan 4 skenario dalam pengujian *SQL Injection* diantaranya pengujian karakter yang memiliki makna dalam *query,* pengujian modifikasi input *login* ketika username diketahui, pengujian modifikasi input *login* ketika username dan password tidak diketahui, dan pengujian injeksi *url*. Dari hasil pengujian dan analisa didapat bahwa serangan *SQL Injection* pada *PHP* dan *Codeigniter* dapat ditangani dengan baik menggunakan fungsi *mysql_real_escape_string*(untuk *PHP*) dan penggunaan *library active record , uri segment*(untuk *Codeigniter*) yang mana dari hasil pengujian didapat bahwa 0% *error query* dan 0% terinjeksi(pada 4 skenario pengujian) dari 104 vektor serangan.

Kata Kunci: Code Igniter, MySQL, PHP, SQL Injection

1. Pendahuluan

Keamanan data adalah hal terpenting didalam sebuah aplikasi web terutama aplikasi web seperti *online banking, online shop,* situs akademik, dan situs-situs pemerintah. Berdasarkan informasi dari OWASP (Open Web Application Security Project) yang telah telah medata sejak tahun 2003 berbagai jenis serangan hacking, dimana SQL Injection masih menduduki peringkat pertama sebagai serangan hacking yang paling sering dilakukan[5], yang mana serangan *SQL Injection* dapat dilakukan melalui input *form* dan *url*[7].

PHP dan Framework Codeigniter merupakan contoh bahasa pemograman yang sering digunakan para developer dalam membangun aplikasi web. Hal tersebut dipengaruhi beberapa faktor diantaranya mudah digunakan, ukuran relatif kecil dan cepat, kemudahan instalasi, terdapat library validasi, dokumentasi lengkap, mendukung PHP 4 dan PHP 5, serta menyediakan banyak fungsi, diantaranya fungsi enkripsi, session, cookies, xss filtering, dan sebagainya[3]. Pada penelitian sebelumnya pernah dilakukan mengenai analisis SQL Injection pada framework Ruby on Rails dan PHP yang mana memaparkan kasus-kasus serangan SQL Injection dan bagaimana ketahanan scripting itu sendiri[2]. Pada penelitian ini, dilakukan analisis bagaimana teknik penanganan SQL Injection pada basis data MySQL dengan framework Codeigniter dan tanpa framework (PHP) sehingga diketahui kemampuan dan *vulnerability* kedua scripting terhadap SQL Injection beserta teknik penanganannya.

ISSN : 2085-4218

2. Dasar Teori

2.1 SQL Injection

SQL Injection dapat diartikan suatu kegiatan menipu *query* dari basis data, sehingga seseorang dapat mengetahui dan mendapatkan informasi yang terdapat pada basis data[9]. Menipu disini adalah menyisipkan kode *SQL* tambahan ke dalam kode yang asli pada sebuah aplikasi. Berikut adalah Ilustrasi *SQL Injection* pada Gambar 1., dimana setelah kode *SQL Injection* disisipkan kedalam perintah *SQL* yang asli. Maka yang akan dieksekusi adalah perintah *SQL Injection*.

Gambar 1. Ilustrasi SQL Injection[1]

Efek yang ditimbulkan dari *SQL Injection* sangat beragam, berikut adalah beberapa *point* bahaya dari serangan tersebut :

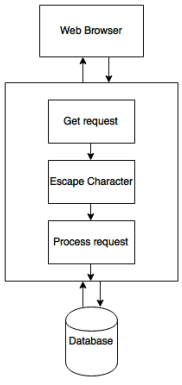
1. *SQL Injection* memungkinkan seseorang dapat *login* (masuk) kedalam sebuah sistem tanpa harus memiliki *account*. Hal ini berlaku baik user biasa maupun admin.
2. Dengan *SQL Injection* pula seseorang bisa menyusup kedalam sebuah basis data sehingga dia dapat merubah, menghapus, atau bahkan menambah data yang terdapat dalam basis data tersebut.
3. Bahkan yang lebih dari sekedar mengubah basis data, sang pelaku bisa menanamkan akun yang tidak diketahui. Sehingga apabila sistemnya telah diperbaiki tapi sang pelaku masih memiliki cadangan akun untuk *login* tanpa harus melakukan *SQL Injection* lagi.

Efek lain dari *SQL Injection* adalah mematikan basis data itu sendiri sehingga *web server* tidak dapat memberikan layanan yang baik kepada para pengguna.

3. Perancangan Sistem

Pada penelitian ini, sistem yang dibangun merupakan bagian dari sebuah aplikasi yaitu fungsi atau kelas yang terdiri dari beberapa method dan disesuaikan dengan jenis *scripting* yang digunakan. Sistem ini pada dasarnya sebagai pemrosesan sebuah input variabel sebelum digabungkan dalam *query sql,* sehingga didapatkan variabel yang aman dari *SQL Injection* sebelum digabungkan kedalam *query*.

3.1 Model Penanganan SQL Injection pada PHP



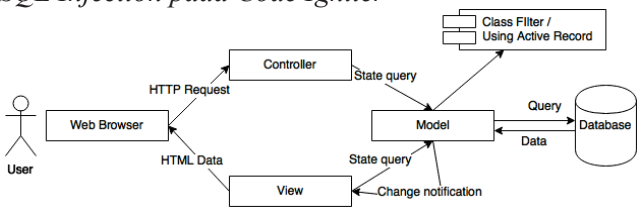
Gambar 2. Model Penanganan SQL Injection pada PHP

Berikut adalah proses penanganan *SQL Injection* pada *PHP* (tanpa *framework*) yang terdapat pada Gambar 2. untuk melakukan pemrosesan variabel input baik dari form input dan *url* pada aplikasi.

ISSN : 2085-4218

1. Get request
   Proses dimana aplikasi akan mengambil request dari user berupa parameter yang berisi input variabel baik dari *url* dan form input. Yang mana parameter tersebut akan berinteraksi dengan database.
2. Escape character
   Kemudian, semua input variabel baik dari url dan form input dilakukan pemrosesan terlebih dahulu menggunakan fungsi *escape character* pada PHP yaitu digunakan fungsi ***mysql_real_escape_string.*** Yang mana fungsi tersebut akan membebaskan karakter-karakter yang dapat menyebabkan keberhasilan *SQL Injection*. Sehingga apabila didalam input variabel baik dari url dan form input terjadi penyisipan perintah *sql*, tidak akan tereksekusi. Karena terbaca sebagai input variabel bukan bagian dari *query sql.*
3. Process Request
   Kemudian input variabel, diproses dan digabung dengan query sql untuk berinteraksi dengan basis data.

3.2 Model Penanganan SQL Injection pada Code Igniter



Gambar 3. Model Penanganan SQL Injection pada Code Igniter

Berikut adalah proses penanganan *SQL Injection* pada Code Igniter, yang mana digunakan *library active record* dalam interaksi ke basis data pada blok model Gambar 3.

1. Controller
   Controller adalah pengontrol semua alur yang bekerja pada Code Igniter, ketika user melakukan *HTTP Request* melalui *browser* akan ditangani oleh *controller*. Kemudian jika dalam proses *request* oleh user terdapat input variabel baik dari *url* ataupun form input yang akan berinteraksi dengan *database*, maka *controller* akan memanggil blok model.
2. Model
   Pada blok model, digunakanlah penggunaan *active record* untuk interaksi ke basis data. *Active record* akan secara otomatis melakukan pembebasan karakter yang memiliki makna dalam query ataupun sintaks *query* yang akan menyebabkan keberhasilan serangan *SQL Injection*. Sehingga ketika terjadi penyisipan query sql pada input variabel , tidak akan tereksekusi sebagai bagian dari query sql. Karena sudah otomatis dilakukan *filtering* oleh *active* record.

3.3 Skenario dan Vektor *Pengujian*

Pengujian yang dilakukan dibagi menjadi 2 yaitu pengujian pada aplikasi PHP dan *Codeigniter* dengan basis data *MySQL* dengan kondisi sebelum dan sesudah adanya penanganan terhadap *SQL Injection* pada masing-masing aplikasi. Hal-hal yang akan dianalisis antara lain :

1. Reaksi masing-masing aplikasi yaitu berbasis *PHP* dan *Codeigniter* ketika menggunakan *database MySQL* apabila dilakukan serangan *SQL Injection* pada kondisi aplikasi belum adanya penanganan terhadap *SQL Injection*.
2. Kemampuan atau *library* atau fungsi pendukung pada aplikasi *PHP* dan *Codeigniter* dalam menangani *SQL Injection*.
3. Reaksi masing-masing aplikasi yaitu berbasis *PHP* dan *Codeigniter* ketika menggunakan basis data *MySQL* apabila dilakukan serangan *SQL Injection* pada kondisi aplikasi sudah adanya penanganan terhadap *SQL Injection*.

Vektor yang digunakan untuk pengujian penyerangan *SQL Injection* pada aplikasi diambil dari beberapa contoh dari literature, *website*, dan hasil modifikasi dari contoh-contoh yang sudah ada. Berikut adalah skenario yang digunakan dalam pengujian serangan *SQL Injection*.

1. Pengecekan terhadap karakter yang memiliki makna dalam *query sql* (18 vektor serangan).

ISSN : 2085-4218

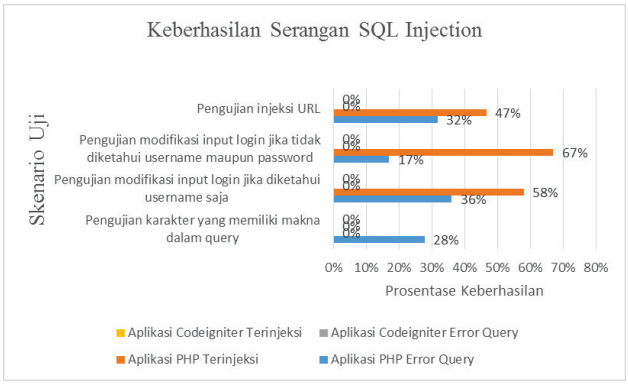
2. Pengujian modifikasi input login jika hanya diketahui username saja (14 vektor serangan).
3. Pengujian modifikasi input login jika tidak diketahui username maupun password (12 vektor serangan).
4. Pengujian injeksi *url*(60 vektor serangan)

4. Hasil Pengujian dan Analisa

Tahap pengujian ini dilakukan pada kedua aplikasi yaitu *PHP* dan *Codeigniter* menggunakan basis data *MySQL* dengan kondisi sebelum dan sesudah adanya penanganan terhadap *SQL Injection*, kemudian dianalisis dari hasil pengujian.

Terdapat kondisi *error query* dan terinjeksi ketika dilakukan pengujian serangan *SQL Injection*. ***Error query*** adalah kondisi dimana terjadi kegagalan perintah eksekusi *query sql* sedangkan **terinjeksi** adalah kondisi dimana perintah *query sql* yang disisipkan melalui aplikasi dapat berjalan namun berhasil dimodifikasi oleh user sehingga didapatkan informasi tertentu seperti dapat melihat, menghapus, mengedit informasi basis data, tabel, data serta melakukan *bypass login* ke aplikasi.

4.1 Hasil Pengujian sebelum adanya Penanganan SQL Injection



Gambar 4. Grafik Keberhasilan Serangan SQL Injection sebelum adanya Penanganan

Gambar 4. merupakan hasil pengujian keberhasilan serangan *SQL Injection* sebelum adanya penanganan pada aplikasi *PHP* dan *Code Igniter*, berikut detail penjelasan hasil pengujian untuk masing-masing vektor pengujian pada aplikasi :

1. Pengujian karakter yang memiliki makna dalam query

Hasil pengujian pengecekan karakter pada aplikasi *PHP* didapat bahwa 28% *error query* , 0% terinjeksi, 72% dapat ditangani dengan baik, dari 18 vektor serangan. Pada kasus ini perlu adanya penanganan terhadap karakter-karakter seperti *single quote*, backslash dan permintaan pengisian semua inputan agar tidak adanya celah untuk serangan sql injection terhadap aplikasi.

Sedangkan hasil pengujian serangan pada aplikasi Codeigniter didapat bahwa 0% *error query,* 0% terinjeksi, 100% dapat ditangani dengan baik, dari 18 vektor serangan**.** Pada pengujian ini aplikasi Codeigniter dapat menangani dengan baik semua karakter yang memiliki makna dalam *query*, dikarenakan penggunaan *active record* secara langsung dilakukan pembebasan karakter yang memiliki makna dalam query. Sehingga pada kasus ini tidak perlu adanya penanganan karena sudah ditangani dengan baik.

2. Pengujian modifikasi inputan jika diketahui username

Hasil pengujian serangan *SQL injection* dengan memodifikasi inputan jika diketahui username pada aplikasi *PHP* didapat bahwa 36% *error query*, 58% terinjeksi, 6% dapat ditangani dengan baik, dari 14 vektor serangan. Berikut adalah beberapa hasil pengujian pada aplikasi PHP yang menyebabkan terjadi injeksi dan dapat melakukan *login* terhadap aplikasi ketika diketahui username saja.

ISSN : 2085-4218

Tabel 1. Pengujian Aplikasi PHP modifikasi input login jika diketahui username saja

No	Vektor		Respon Aplikasi
	Username	Password	
1	admin'/*	*/--‘	Terjadi injeksi , berhasil login
2	admin	1‘or ’1’ = ’1	Terjadi injeksi , berhasil login
3	admin	1‘ or 1=1#	Terjadi injeksi , berhasil login

Pada kasus ini perlu dilakukan pemrosesan input variabel dari username dan password sehingga input yang diberikan oleh user benar-benar aman dan tidak bisa memodifikasi apapun *query* aslinya serta diberikan permintaan pengisian input username dan password.
Sedangkan hasil pengujian serangan pada aplikasi *codeigniter* didapat bahwa 0% *error query*, 0% terinjeksi, 100% dapat ditangani dengan baik, dari 14 vektor serangan**.** Pada kasus ini, semua modifikasi inputan tidak berhasil menembus dan dapat ditangani dengan baik oleh *active record* pada codeigniter.

3. Pengujian memodifikasi inputan jika tidak diketahui username dan password

Hasil pengujian serangan *SQL injection* dengan memodifikasi inputan jika tidak diketahui username maupun password pada aplikasi *PHP* didapat bahwa 17% *error query*, 67% terinjeksi, 16% dapat ditangani dengan baik, dari 12 vektor serangan. Berikut adalah beberapa hasil pengujian hasil modifikasi inputan yang menyebabkan terjadi injeksi dan dapat melakukan *login* terhadap aplikasi tanpa diketahui username maupun password.

Tabel 2. Pengujian Aplikasi PHP memodifikasi input login jika tidak diketahui username dan password

No	Vektor		Respon Aplikasi
	Username	Password	
1	Test	‘ or 1=1--‘	Terjadi injeksi, berhasil login
2	Test	‘or username like ‘%a%	Terjadi injeksi, berhasil login

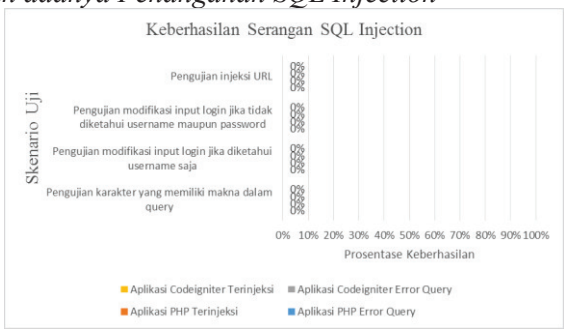
Pada pengujian memodifikasi *form* input jika tidak diketahui username dan password pada aplikasi *PHP* perlu adanya penanganan untuk permintaan setiap input ketika kosong dan pembebasan karakter yang memiliki makna dalam *query* yang dapat memodifikasi *query sql* asli. Sedangkan hasil pengujian serangan pada aplikasi *codeigniter* didapat bahwa 0% *error query*, 0% terinjeksi, 100% dapat ditangani dengan baik, dari 12 vektor serangan**.** Pada kasus ini, semua modifikasi inputan tidak berhasil menembus dan dapat ditangani dengan baik oleh *active record* pada codeigniter.

4. Pengujian dengan injeksi URL dengan tipe data *integer* dan *string*

Hasil pengujian serangan *SQL Injection* pada aplikasi *PHP* melalui injeksi *URL* dengan tipe data *integer* dan *string* didapat bahwa 32% *error query*, 47% terinjeksi, 21% ditangani dengan baik, dari 60 vektor serangan. Terdapat beberapa kondisi yang berhasil menginjeksi *database* yang mana dengan mudah sekali *malicious user* menyisipkan query sql melalui parameter GET pada *url* aplikasi PHP jika pada aplikasi belum adanya pembatasan input. Kondisi ini terjadi karena sintaks yang disisipkan tidak dibaca atau dianggap sebagai satu kesatuan nilai yang dikirim pada *url* seperti http://localhost/testing_php/?page=news&id_news=2’. Tanda *single quote* yang disisipkan memunculkan error, merupakan suatu celah bahwa karakter *single quote* dieksekusi oleh basis data bukan bagian dari nilai yang dikirim, maka dari itu dengan mudah dapat dilakukan penyisipan query yang lain. Sehingga pada kasus ini perlu adanya penanganan pada parameter yang dikirim melalui *url* supaya didapatkan nilai yang aman ketika digabung dengan sintaks sql.
Sedangkan pengujian pada aplikasi Codeigniter didapat bahwa 0% error query, 0% terinjeksi, 100% dapat ditangani dengan baik, dari 60 vektor serangan. Penggunaan URI dalam *url* codeigniter dengan baik dapat menangani secara langsung apabila terdapat karakter-karakter yang seharusnya tidak boleh diinputkan, ketika dilakukan manipulasi *url* tidak satupun yang berhasil. Karena dalam URI hanya karakter-karakter tertentu yang diizinkan sehingga apabila dimasukan karakter yang tidak diizinkan akan dibebaskan oleh *library uri* code igniter.

ISSN : 2085-4218

4.2 Hasil Pengujian sesudah adanya Penanganan SQL Injection



Gambar 4. Grafik Keberhasilan Serangan SQL Injection sesudah adanya Penanganan

Seperti yang didapat pada Gambar 4., sesudah adanya penanganan menggunakan fungsi maupun *library* pada aplikasi PHP(model penanganan terdapat pada gambar 2.**)** dan Code Igniter(model penanganan terdapat pada gambar 3.) bahwa kedua aplikasi dapat menangani dengan baik adanya serangan *SQL Injection*. Yang mana pada kedua aplikasi PHP dan Code Igniter dapat menangani dengan baik semua serangan pengujian *SQL Injection*.

5. Kesimpulan

Serangan *SQL Injection* pada aplikasi PHP dapat ditangani dengan baik dengan menerapkan fungsi *mysql_real_escape_string* untuk *filtering* input variabel namun penggunaan fungsi tersebut akan menambahkan karakter *backslash*(\) didepan karakter *single quote*, double quote, dan *backslash*.

Sedangkan serangan *SQL Injection* pada aplikasi Code Igniter dapat ditangani dengan baik dengan menerapkan penggunaan *active record* dalam interaksi ke *database,* namun penggunaan *active record* tersebut tidak akan menambahkan karakter apapun pada input variabel seperti pada aplikasi PHP.

Untuk penelitian selanjutnya dapat dilakukan analisis pengaruh penggunaan *active record* dan *native query* pada Code Igniter untuk interaksi ke *database*, bisa dilihat dari kecepatan eksekusi *query* dan program, keamanan, serta dukungan terhadap jenis *database* lain seperti *Oracle, Mongo DB, SQL Server,* dan sebagainya.

6. Daftar Referensi

[1] Clarke, Justin, *SQL Injection Attack and Defense*, USA: Elsevier, 2012.
[2] Damarisa, Febriana, "Analisis dan Perbandingan SQL Injection pada PHP dan Ruby on Rails", Universitas Telkom, Bandung, 2012.
[3] Daqiqil Id, Ibnu, *Framework Codeigniter*, Pekanbaru, 2011.
[4] Jasman, Web Hacking dari Subkultural Programmer, Yogyakarta:Andi, 2015.
[5] (2015) OWASP Website. [Online]. Availabale: https://www.owasp.org/index.php/Top_10_2013-Top_10
[6] Saputra, Agus, Trik dan Solusi Jitu Pemrograman PHP, Jakarta:Lokomedia, 2011.
[7] Sharma, Chandershekhar and Dr.S.C, Jain, *Analysis and Classification of SQL Injection Vulnerabilities and Attack on Web Applications*, IEEE, 2014.
[8] XuePing-Chen, *SQL Injection Attack and guard technical Research*, 2011.
[9] Zam, Efvy, Teknik Hacking dengan SQL Injection.Jakarta: PT Elex Media Komputindo, 2014.