Mencegah SQL *Injection* Pada Sebuah *Website* Menggunakan *Pattern* pada *Regex*

Luthfi Hamid Masykuri - 13512100[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512100@std.stei.itb.ac.id

***Abstrak*— Makalah ini membahas tentang metode pencegahan SQL Injection. Metode yang digunakan adalah validasi Query SQL menggunakan Regex. Di dalam makalah ini akan dibahas bagaimana SQL injection bekerja. Serta pencegahannya menggunakan Regex. Akan dibahas pula apa saja pattern Regex yang digunakan. Selanjutnya akan diimplementasi dalam bentuk fungsi program dalam Bahasa pemrograman.**

***Kata Kunci*— hacking,sql,injection,website.**

I. PENDAHULUAN

Pada masa sekarang ini teknologi informasi mengalami kemajuan yang sangat pesat. Bahkan masa sekarang ini disebut jaman informasi. Di jaman yang serba informasi ini pastinya kita sudah akrab dengan sesuatu yang bernama *website.*

Sudah banyak *website* yang ada saat ini. Ribuan bahkan jutaan dengan konten layanan yang banyak pula. Konten-konten ini tentunya menympan berbagai data. Data pada *website* biasanya disimpan dalam bentuk *database.*

Biasanya *Database Management System* menggunakan *SQL (Syntax Query Languange)*. *SQL* merupakan Bahasa *DBMS* yang lazim dipakai para *web programmer* dalam menyimpan data-data pada *website.*

Akan tetapi penggunaan ini kadang menimbulkan suatu celah. Celah ini muncul dari *form* atau *request* pada aplikasi *website.* Hal ini bias menarik perhatian para *hacker* untuk berbuat *jahil.*

Dalam dunia *hacking,* hal ini lazim disebut dengan *SQL Injection.* Metode ini sering dipakai para *hacker* untuk memasuki suatu system *website.* Bahkan bias lebih dari itu, tangan *jahil* yang beruntung bias memasuki jaringan dalam *website* tersebut serta memanipulasi atau mencuri data-data yang ada dalam jaringan tersebut.

Bayangkan jika yang dibobol adalah sebuah layanan yang menyimpan data penting kita. Misalnya bank tempat kita menaruh sejumlah uang kita. Tangan *jahil* beruntung tersebut mungkin akan membobol rekening kita.

Tentunya kita tidak ingin hal itu terjadi. Oleh karena itu dalam makalah ini penulis akan membahas cara untuk mencegah *SQL Injection.* Metode yang akan digunakan penulis adalah pencocokan string menggunakan *Regex (Regular Expression).*

Dalam makalah ini penulis juga akan membahas tentang cara kerja *SQL Injection* dalam sebuah *website* serta pengahannya dengan Regex.

II. LANDASAN TEORI

A. SQL (Structured Query Languange)

SQL adalah sebuah bahasa yang digunakan dalam implementasi terhadap data yang disimpan pada sebuah basis data (*database).* Seperti halnya dalam dalam pembuatan aplikasi pada komputer bahasa pemrograman dugunakan untuk membangunnya. Begitu juga untuk pembuatan *database,* digunakan bahasa untuk membangun suatu *database.*

Bahasa *SQL* terdiri dari sekumpulan perintah-perintah yang digunakan untuk mengakses data pada sebuah database relasional. *SQL* merupakan bahasa baku untuk sebuah *Relational Database Management System (RDBMS)* yang saat ini hampir digunakan oleh semua *Database Server* untuk melakukan manajemen datanya. Bahasa *SQL* Sendiri telah diresmikan sebagai bahasa computer yang mengikuti standar *ANSI (American National Standard Institute)* sebagai bahasa dalam manajemen basis data relasional.

Sejarah *SQL* awalnya pada tahun 1970 dirancang oleh peneliti dari IBM yang bernama EF Codd dengan artikelnya yang membahas tentang ide-ide untuk pembuatan basis data relasional. Dalam artikel tersebut juga membahas kemungkinan penggunaan bahasa standar untuk mengakses sebuah basis data untuk menjalankan fungsi *database.* Kemudian tercetuslah *SEQUEL (Structured English Query Languange)* sebagai nama bahasa dalam relasional tersebut. Akan tetapi, karena permasalahan hukum mengenai penamaan SEQUEL, IBM pun mengubahnya menjadi SQL.

Di akhir tahun 1970-an, muncul perusahaan bernama Oracle yang membuat server basis data populer yang bernama sama dengan nama perusahaannya. Dengan naiknya kepopuleran Oracle, maka SQL juga ikut populer sehingga saat ini menjadi standar de facto bahasa dalam manajemen basis data. Standarisasi SQL dimulai pada tahun 1986, ditandai dengan dikeluarkannya standar SQL oleh ANSI. Standar ini sering disebut dengan SQL86.Standar tersebut kemudian diperbaiki pada tahun 1989 kemudian diperbaiki lagi pada tahun 1992. Versi

terakhir dikenal dengan SQL92.

Pada tahun 1999 dikeluarkan standar baru yaitu SQL99 atau disebut juga SQL99, akan tetapi kebanyakan implementasi mereferensi pada SQL92. Saat ini sebenarnya tidak ada server basis data yang 100% mendukung SQL92. Hal ini disebabkan masing-masing server memiliki dialek masing-masing. Secara umum, SQL terdiri dari dua bahasa, yaitu *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML). Implementasi DDL dan DML berbeda untuk tiap sistem manajemen basis data (SMBD), namun secara umum implementasi tiap bahasa ini memiliki bentuk standar yang ditetapkan ANSI.

B. SQL Injection

SQL injection adalah jenis aksi hacking pada keamanan komputer di mana seorang penyerang bisa mendapatkan akses ke basis data di dalam sistem. *SQL* Injection yaitu serangan yang mirip dengan serangan *XSS* dalam bahwa penyerang memanfaatkan aplikasi vektor dan juga dengan Common dalam serangan *XSS*.

SQL injection merupakan salah satu kelemahan yang paling dahsyat untuk dampak bisnis, karena dapat menyebabkan pembongkaran semua informasi yang sensitif yang tersimpan dalam sebuah aplikasi database, termasuk informasi berguna seperti username, password, nama, alamat, nomor telepon, dan rincian kartu kredit. Jadi *SQL* injection adalah kelemahan yang diterjadi ketika penyerang mampu mengubah *Structured Query Language (SQL)* di dalam *database*. Dengan mempengaruhi *database*, penyerang dapat memanfaatkan sintaks dan kemampuan dari SQL itu sendiri, serta kekuatan dan fleksibilitas yang mendukung fungsi database dan fungsi sistem operasi yang hanya dilakukan dalam *database*.

SQL Injection adalah serangan dimana kode *SQL* dimasukan atau ditambahkan ke dalam aplikasi atau user input parameter yang kemudian akan diteruskan ke *SQL Server* untuk *parsing* dan eksekusi. Setiap prosedur yang membangun pernyataan *SQL* berpotensi untuk mudah terserang.

Bentuk injeksi ini utamanya berbentuk penyisipan langsung kode ke dalam parameter input yang digabungkan dengan perintah *SQL* dan kemudian dieksekusi. Sebuah serangan langsung paling sedikit menyuntikan kode ke dalam string yang ditujukan ke penyimpanan di dalam tabel atau sebagai metadata. Ketika string yang tersimpan akhirnya digabungkan ke dalam perintah *SQL* dinamis, kode tersebut akan dieksekusi. Ketika aplikasi web gagal untuk membersihkan paramater dengan benar dilewatkan ke pernyataan *SQL* yang dibuat secara dinamis (bahkan ketika menggunakan teknik parameterisasi) ini memungkinkan untuk seorang penyerang untuk mengubah mengubah konstruksi back-end *SQL*.

Saat seorang *hacker* dapat memodifikasi perintah *SQL*, perintah tersebut akan tereksekusi dengan hak yang sama dengan aplikasi pengguna, saat menggunakan *server SQL* untuk mengeksekusi perintah yang berinteraksi dengan system operasi, proses akan berjalan dengan hak akses yang sama dengan komponen yang mengeksekusi perintah. Dalam kasus ini contohnya adalah *database server*, aplikasi *server*, atau *web server*.

C. Regular Expression (Regex)

Regular Expression (Regex) adalah suatu *pattern matching*, yaitu notasi yang digunakan untuk mencocokan teks dan data, serta memanipulasinya. *Regex* mempunyai 2 fungsi utama yaitu mencari dan mengganti.

Regex digunakan oleh banyak teks editor, *utilities*, dan bahasa pemrograman untuk pencarian dan memanipulasi teks berdasarkan pola. Misalnya, *Perl, Ruby, PHP, VB* dan *Tcl* memiliki *engine* Regex yang kuat dibangun pada *syntax* mereka.

Sejarah *Regex* dimulai pada tahun 1950an oleh Stephen Cole Kleene seorang ahli matematika membuat sebuah model pencocokan string menggunakan notasi matematika yang disebut *regular sets.* Ken Thompson mengimplementasikan noatasi tersebut ke dalam editor buatannya, *QED*, untuk pencarian string dengan pola tertentu. Dia juga menambahkan fitur tersebut ke dalam *ed*, sebuah editor teks dalam sistem operasi *unix*. Untuk pencarian string dalam *ed* menggunakan pattern, perintahnya : /g /re /p (/g : globally, /re : regular expression, /p : print) yang maksudnya adalah pencarian global baris-baris dalam sebuah file yang memiliki pola tertentu, dan ditampilkan atau dicetak. Istilah grep akhirnya digunakan sebagai sebuah nama sebuah program atau tool dalam system operasi unix untuk fungsi yang sama yaitu pencarian string.

Engine *Regex* terdiri dari 2 jenis *Text-directed engine* dan *regex-directed* atau ada juga yang menyebut *DFA (Deterministic Finite Automaton)* dan *NFA (Nondeterministic Finite Automatan) engines.* Namun jenis yang lebih banyak diminati adalah *regex-directed engine*, disamping itu *feature*-nya lebih hebat dari *tex-directed engine.*

Misalnya saja kita mencocokan kata “regex” dengan kata “belajar regex sekarang”, pertama-tama regex akan memulai mencocokan karakter pertama yaitu huruf “R” dengan “B”, karena tidak cocok maka akan dilanjutkan ke tempat selanjutny yaitu “R” dengan “E”, karena masih belum cocok juga proses ini akan terus berlanjut, hingga akhirnya menemukan hurup “R” yaitu di posisi ke-7, hasil ini akan disimpan di memori bahwa telah valid pada posisi ke-7, kemudian akan meneruskan langkah selanjutnya mencocokan hurup “E” dengan ” ” dan ternyata tidak cocok, engine akan mulai melakukan pencocokan kembali. Karena huruf “E” tidak ditemukan maka kali ini engine akan kembali menggunakan huruf “R” dan dicocokan pada posisi ke-9 yakni huruf “R”. Karena valid, engine menggunakan huruf berikutnya yakni huruf “E” dan dicocokan dengan posisi ke-10 yakni huruf=”E”. Proses ini akan terus diulang hingga mencapai posisi akhir suatu himpunan string. Nah jika kata “regex” ditemukan oleh kata “belajar regex sekarang” maka engine akan melaporkan bahwa Regular Expression telah valid, meskipun masih ada karakter yang belum di validasi (kata “sekarang”). Itu adalah cara kerja NFA yang lama, berbeda dengan mesin NFA yang sekarang, sudah lebih baik, yaitu meskipun string yang dicari telah match tetap akan diteruskan sampai benar-benar valid ditemukan. Setelah itu

baru dibuat laporan sukses. Berbeda dengan engine dari NFA, engine DFA memilik cara kerja yakni membandingkan semua karakter secara serempak, hal ini membuat total memory yang dibutuhkan lebih besar dari NFA, sehingga teknologi NFA lebih diminati.

Fungsi *regex* salah satunya bias didefinisikan sebagai berikut :

$$\emptyset\ Regex(Patern, Text)$$

Fungsi ini akan menghasilkan *TRUE*, jika ternyata pola cocok dengan sumber dan *FALSE* jika keduanya tidak cocok. Cara membuat rumus ini adalah dengan menggunakan rumus meta. Meta adalah karakter yang memiliki arti tertentu. Contoh pada perintah DOS adalah DIR *.*, karakter * tidak diartikan sebagai karakter bintang oleh komputer melainkan mempunyai arti khusus.

Ada meta karakter khusus lainnya, yaitu karakter meta |. Karakter meta ini menjadikan karakter meta text sebagai pilihan pada pola untuk pencocokan dengan sumber. Berikut adalah pola yang umum terdapat pada *Regex*.

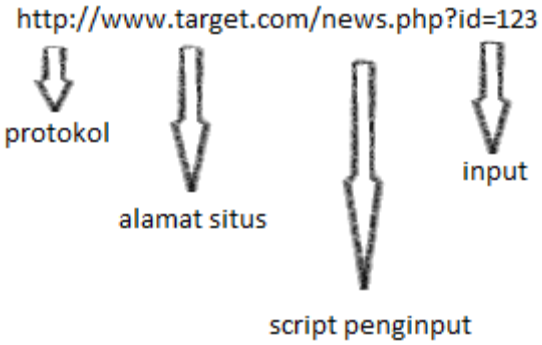
Pola	Penjelasan
[ ]	ekspresi kurung. cocok dengan satu karakter yang berada dalam kurung, misal: pattern "a[bcd]i" cocok dengan string "abi", "aci", dan "adi". penggunaan range huruf dalam kurung diperbolehkan, misal pattern "[a-z]" cocok dengan salah satu karakter diantara string "a" sampai "z". pattern [0-9] cocok dengan salah satu angka. jika ingin mencari karakter "-" juga, karakter tersebut harus diletakkan di depan atau di belakang kelompok, misal: "[abc-]".
[^]	cocok dengan sebuah karakter yang tidak ada dalam kurung, berlawanan dengan yang diatas. misal: pattern "[^abc]" cocok dengan satu karakter apa saja kecuali "a", "b", "c".
?	cocok dengan nol atau satu karakter sebelumnya. misal: pattern "died?" cocok dengan string "die" dan "died".
+	cocok dengan satu atau lebih karakter sebelumnya. misal: "yu+k" cocok dengan "yuk", "yuuk", "yuuuk", dan seterusnya.
*	cocok dengan nol atau lebih karakter sebelumnya. misal: pattern "hu*p" cocok dengan string "hp", "hup", "huup" dan seterusnya.
{x}	cocok dengan karakter sebelumnya sejumlah x karakter. misal: pattern "[0-9]{3}" cocok dengan bilangan berapa saja yang berukuran 3 digit.
{x,y}	cocok dengan karakter sebelumnya sejumlah x hingga y karakter. misal: pattern "[a-z]{3,5}" cocok dengan semua susunan huruf kecil yang terdiri dari 3 sampai 5 huruf.
!	jika diletakkan di depan pattern, maka berarti "bukan". misal pattern "!a.u" cocok dengan string apa saja kecuali string "alu", "anu", "abu", "asu", "aiu", dan seterusnya .
^	jika diletakkan di depan pattern, akan cocok dengan awal sebuah string.
( )	gruping. digunakan untuk mengelompokkan karakter - karakter menjadi single unit. string yang cocok dalan pattern yang berada dalam tanda kurung dapat digunakan pada operasi berikutnya. semacam variable.
\	escape character. mengembalikan fungsi metacharacter menjadi karakter biasa. pada beberapa system dapat berarti sebaliknya, yaitu metacharacter menggunakan escape character didepannya .
$	jika diletakkan di belakang pattern, akan cocok dengan akhir sebuah string.

Selain karakter *meta* di atas PHP juga menyediakan class karakter yang fungsinya sama dengan meta, beberapa class di PHP tersebut yaitu, [[:alpa]] akan berarti sembarang huruf, [[:digit]] akan mengwakilik angka, [[[:alnum]] akan berarti angka atau digit, [[:space]] akan berarti sembarang space, [[:upper]] akan berarti sembarang huruf kapital dan [[:lower]] akan berarti sembarang huruf kecil.

III. PEMBAHASAN

A. SQL Injection

Pada bagian ini penulis akan membahas bagaimana *SQL Injection* bekerja pada sebuah website. Pertama kita akan menetapkan target. Kita akan mengetes *vulnerabilitas* dari target tersebut. Paling mudah adalah cari alamat yang mengandung input *request*. Biasanya berupa seperti di bawah berikut.



Asumsikan script news.php diatas memiliki script *SQL* yang berisi sebagai berikut

```
<?php
```

$id = $_GET[‘id’];

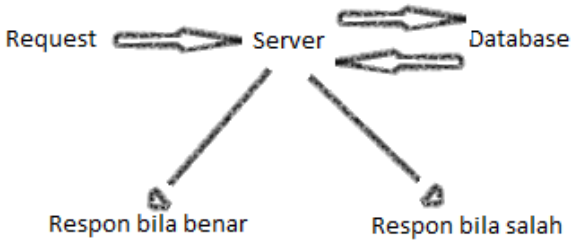
$query = “SELECT * FROM news WHERE id = ‘$id’”;

?>

Kode diatas sangat rawan. Untuk mengetesnya kita coba menambahkan karakter ‘. Sehingga alamatnya menjadi http://www.target.com/news.php?id=123’. Parameter input tersebut akan menyebabkan error. Karena kode *SQL* yang dieksekusi menjadi seperti ini.

SELECT * FROM news WHERE id = ‘123’’;

Tentu sintaks diatas akan menyebabkan error. Karena ada tanda petik yang seharusnya pada *SQL* jika sebagai input harus dengan *escape (‘\’) character*. Kemungkinan proses pada server dapat digambarkan pada skema berikut ini.



Error tersebut mengindikasikan adanya kesalahan input yang diproses. Saat input yang diberikan 123, server akan melakukan cek pada *database* sesuai, sehingga server akan memberikan respon benar berupa tampilan awal website tersebut. Namun berbeda halnya saat anda memasukan tanda petik pada input. Server tidak menemukan kecocokan input dengan *database* sehingga server menampilkan pesan error. Dari situ lah bias disimpulkan bahwa website tersebut memungkinkan untuk disisipi perintah *SQL* melalui parameter input.

Selanjutnya kita harus menentukan jumlah kolom pada table yang berada di dalam *database* target. Penentuan jumlah kolom ini perlu dilakukan karena kita perlu mengetahui kolom mana dari sebuah tabel yang bisa dimanfaatkan. Hal ini bertujuan supaya kita bisa memasukkan perintah SQL Injection pada lokasi yang tepat. Sebab kalau kita memasukkan di tempat yang salah maka kita tidak akan memperoleh apapun. Untuk melakukan hal ini agak bersifat *“trial and error”*, dimana perintah yang digunakan adalah ORDER BY.

Untuk mudahnya berikut adalah daftar berbagai *Sysntax* untuk *SQL Injection*

1. *Commention Out*

   Gunanya untuk mengakhiri suatu query, bypass query.

   + SQL Server

   Syntax: –

   Penggunaan: DROP namatabel;–

   + MySQL

   Syntax: #

   Penggunaan: DROP namatabel;#

   Contoh penggunaan :

   * Username: admin’–

   * Proses query yang terjadi di server:

   SELECT * FROM userlist WHERE username=’admin’–’ AND password=’password’;

   Query ini akan memberikan km akses sebagai admin karena query selanjutnya setelah — akan diabaikan

2. *Inline Comment*

   Gunanya untuk mengetahui versi SQL server yang digunakan atau untuk bypass script proteksi

   + SQL Server (MySQL juga bisa)

   Syntax: /*Comment*/

   Penggunaan: DROP/*comment*/namatabel

   atau: DR/**/OP/*bypass proteksi*/namatabel

   atau: SELECT/*menghindari-spasi*/password/**/FROM/**/userlist

   + MySQL (mendeteksi versi)

   Syntax: /*!MYSQL Special SQL*/

   Penggunaan: SELECT /*!32302 1/0,*/1 FROM namatabel

   Note: Syntax juga bisa digunakan jika versi MySQL lebih tinggi dari 3.23.02 (sesuai query), tidak berfungsi untuk versi dibawahnya

3. *Staking Queries*

   Gunanya untuk menyambung 2 buah query dalam 1 transaksi.

   + SQL Server

   Syntax: ;

   Penggunaan: SELECT * FROM namatabel; DROP namatabel–

4. Pernyataan IF

   Ini kunci jika melakukan Blind SQL Injection, juga berguna untuk testing sesuatu yang ga jelas secara akurat

   + SQL Server

   Syntax: IF kondisi bagian-true ELSE bagian-false

   Penggunaan: IF (1=1) SELECT ‘true’ ELSE SELECT ‘false’

+ MySQL

Syntax: IF(kondisi,bagian-true,bagian-false)

Penggunaan: SELECT IF(1=1,'true','false')

5. Operasi String

Gunanya untuk bypass proteksi

+ SQL Server

Syntax: +

Penggunaan: SELECT login + '-' + password FROM userlist

+ MySQL Server

Syntax: ||

Penggunaan: SELECT login || '-' || password FROM userlist

6. *Union Injection*

Gunanya menggabungkan 2 tabel yang berbeda dengan syarat tabel itu harus sama jumlah kolomnya.

Penggunaan: ' UNION SELECT * FROM namatabel

atau: ' UNION ALL SELECT * FROM namatabel

atau: ' UNION SELECT kolom1,kolom2 FROM namatabel

Proses yang terjadi dalam query:

SELECT * FROM user WHERE id=' 1′ UNION SELECT kolom1,kolom2 FROM namatabel

B. Validasi Query SQL dengan Regex

Dalam validasi ini akan ditentukan strategi dalam menggunakan *Regex*. Kita akan menggunakan beberapa *Regex* untuk *parsing* dan validasi *statement* dalam query *SQL*. Berikut ada tiga strategi *Regex* yang akan digunakan.

1. *Regular Expression* untuk identifikasi *text blocks.*
2. *Regular Expression* untuk identifikasi *statement breaks.*
3. *Regular Expression* untuk identifikasi *SQL Statement.*

Pertama kita akan membahas *Text Blocks.* Intinya di sini menghilangkan seperti *multi-row comment.* Berikut adalah *Pattern Regex* untuk *Text Blocks.*

Pattern = '('|[^']*'

Selanjutnya adalah *Regex* untuk *Statement Break.* Pada tahap ini intinya kita akan akan mengecek adanya semicolon (;) pada *Query.*

Kemudian kita membutuhkan ekspresi untuk menemukan *SQL Statements.* Kata-kata yang perlu dicari dalam *Query* ini antara lain *ALTER, CREATE, DELETE, DROP, EXEC, EXECUTE, INSERT, INSERT INTO, MERGE, SELECT, UPDATE, UNION,atau UNIAL ALL.* Sehingga *Pattern* yang kita butuhkan sebagai berikut.

```
\b(ALTER|CREATE|DELETE|DROP|EXEC(UTE){
0,1}|INSERT(
+INTO){0,1}|MERGE|SELECT|UPDATE|UNION(
+ALL){0,1})\b
```

Selanjutnya kita perlu membuat Algoritma untuk Validatornya. Pertama kita perlu membuat Kelas *StatementTypes* yang atributnya berupa *Statement* dalam *SQL*. Berikut adalah Kelas *StatementTypes.*

```
Public Enum StatementTypes
  None = 0
  Procedure = 0
  Alter = 1
  Create = 2
  Delete = 4
  Drop = 8
  Execute = 16
  Insert = 32
  [Select] = 64
  Update = 128
  Union = 256
  Batch = 512
  Merge = 1024 Or Delete Or Insert Or
[Select] Or Update
End Enum
```

Kemudian Berikut adalah algoritama untuk Validator yang yang akan menggunakan Kelas *StatementTypes* juga.

```
Public Class CommandTextValidator
  Public Shared Sub
ValidateStatement(ByVal commandText As
String, ByVal authorizedStatements As
StatementTypes)
  'Construct Regular Expression To
Find Text Blocks, Statement Breaks &
SQL Statement Headers
  Dim regExText As String =
"('(''|[^'])*')|(;)|(\b(ALTER|CREATE|D
ELETE|DROP|EXEC(UTE){0,1}|INSERT( +INT
O){0,1}|MERGE|SELECT|UPDATE|UNION( +AL
L){0,1})\b)"

  'Remove Authorized Options
  If (authorizedStatements And
StatementTypes.Batch) =
StatementTypes.Batch Then regExText =
regExText.Replace("(;)", String.Empty)
  If (authorizedStatements And
StatementTypes.Alter) =
StatementTypes.Alter Then regExText =
regExText.Replace("ALTER",
String.Empty)
  If (authorizedStatements And
StatementTypes.Create) =
```

```
StatementTypes.Create Then regExText =
regExText.Replace("CREATE",
String.Empty)
  If (authorizedStatements And
StatementTypes.Delete) =
StatementTypes.Delete Then regExText =
regExText.Replace("DELETE",
String.Empty)
  If (authorizedStatements And
StatementTypes.Delete) =
StatementTypes.Delete Then regExText =
regExText.Replace("DELETETREE",
String.Empty)
  If (authorizedStatements And
StatementTypes.Drop) =
StatementTypes.Drop Then regExText =
regExText.Replace("DROP",
String.Empty)
  If (authorizedStatements And
StatementTypes.Execute) =
StatementTypes.Execute Then regExText
= regExText.Replace("EXEC(UTE){0,1}",
String.Empty)
  If (authorizedStatements And
StatementTypes.Insert) =
StatementTypes.Insert Then regExText =
regExText.Replace("INSERT( +INTO){0,1}
", String.Empty)
  If (authorizedStatements And
StatementTypes.Merge) =
StatementTypes.Merge Then regExText =
regExText.Replace("MERGE",
String.Empty)
  If (authorizedStatements And
StatementTypes.Select) =
StatementTypes.Select Then regExText =
regExText.Replace("SELECT",
String.Empty)
  If (authorizedStatements And
StatementTypes.Union) =
StatementTypes.Union Then regExText =
regExText.Replace("UNION",
String.Empty)
  If (authorizedStatements And
StatementTypes.Update) =
StatementTypes.Update Then regExText =
regExText.Replace("UPDATE",
String.Empty)

  'Remove extra separators
  Dim regExOptions As RegexOptions =
regExOptions.IgnoreCase Or
regExOptions.Multiline
  regExText = Regex.Replace(regExText,
"\(\|", "(", regExOptions)
  regExText = Regex.Replace(regExText,
"\|{2,}", "|", regExOptions)
  regExText = Regex.Replace(regExText,
"\|\)", ")", regExOptions)

  'Check for errors
  Dim patternMatchList As
MatchCollection =
Regex.Matches(testText, regExText,
regExOptions)
  For patternIndex As Integer =
patternMatchList.Count - 1 To 0 Step -
1
    Dim value As String =
patternMatchList.Item(patternIndex).Va
lue.Trim
    If
String.IsNullOrWhiteSpace(value) Then
      'Continue - Not an error.
    ElseIf value.StartsWith("'")
AndAlso value.EndsWith("'") Then
      'Continue - Text Block
    ElseIf value.Trim = ";" Then
      Throw New
System.UnauthorizedAccessException("Ba
tch statements not authorized:" &
vbCrLf & commandText)
    Else
      Throw New
System.UnauthorizedAccessException(val
ue.Substring(0, 1).ToUpper &
value.Substring(1).ToLower & "
statements not authorized:" & vbCrLf &
commandText)
    End If
  Next
 End Sub
End Class
```

IV. PERMASALAHAN

Adanya validasi untuk mencegah tindakan *hacking* melalui *SQL injection* tentunya menguntungkan. Akan tetapi akan muncul masalah baru. Karena setiap *Query* harus divalidasi, sehingga tentunya akan menambah waktu proses dalam mengirim perintah. Perlu adanya algoritma yang mangkus untuk validator ini.

V. KESIMPULAN

Penggunaan *String Matching* dengan *Regex* ternyata mampu untuk mencegah *SQL Injection* untuk masuk ke dalam *System* aplikasi yang kita buat, terutama yang berhubungan dengan *Database SQL*.

VI. UCAPAN TERIMA KASIH

Alhamdulillah, saya ucapkan atas rahmat Allah SWT sehingga saya bisa menyelesaikan makalah ini. Tak lupa juga saya ucapkan terima kasih kepada orang tua tercinta. Terima kasih juga saya ucapkan kepada para dosen Labtek V tercinta dan juga para teman-teman saya yang selalu sedia membantu saya ketika kesulitan.

REFERENCES

[1] Eka, Dhiah, "REGULAR EXPRESSION(RegEx) PADA C#", 2013, http://dhiaheka.blogspot.sg/2013/05/regular-expressionregex-pada-c_2196.html Diakses pada Senin, 19 Mei 2014 jam 03:47

[2] Fajar, Agus, dkk. "SQL Injection", 2013, http://anak-takalar.blogspot.sg/2013/10/makalah-sql-injection.html Diakses pada Minggu 18 Mei 2014 jam 19:23

[3] Widigdha, Aryya Dwisatya. "7 Hari Menjadi Jawara di Internet", 2012, Yogyakarta : Penerbit Gava Media.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2014

Luthfi Hamid Masykuri
13512100