SQL injections

Current Web Environment

- Current "Web pages" are really Web applications
  - Front-end which may run in browser
  - Server provides execution environment
  - Back-end which provides services
  - Database for persistent storage
- Interfaces connect the different subsystems
  - E.g. HTTP, REST, WebSocket, SQL, etc..
- Multiple technologies and languages used
  - E.g. Javascript, PHP, HTML, CSS

Current Web Environment

- Each subsystem may be vulnerable to attacks
  - Entire application may be compromised if single breach is found
- SQL Injections are just one case
  - Focus in applications using SQL servers
  - There are many other attacks

What?

- Conjunction of several things:
  - Specially crafted input
  - Lack of sanity checks in code
- Injection of an SQL statement into another SQL statement
  - Changing its original purpose
- Most frequent vector: attacker injects special SQL statement into text field

SQL injection

Form provides two values: login and password

Typical validation query:

```
SELECT user FROM users WHERE user='$login' AND password='$password'
```

For login=admin and password=1234, query becomes:

```
SELECT user FROM users WHERE user='admin' AND password='1234'
```

SQL injection: detection

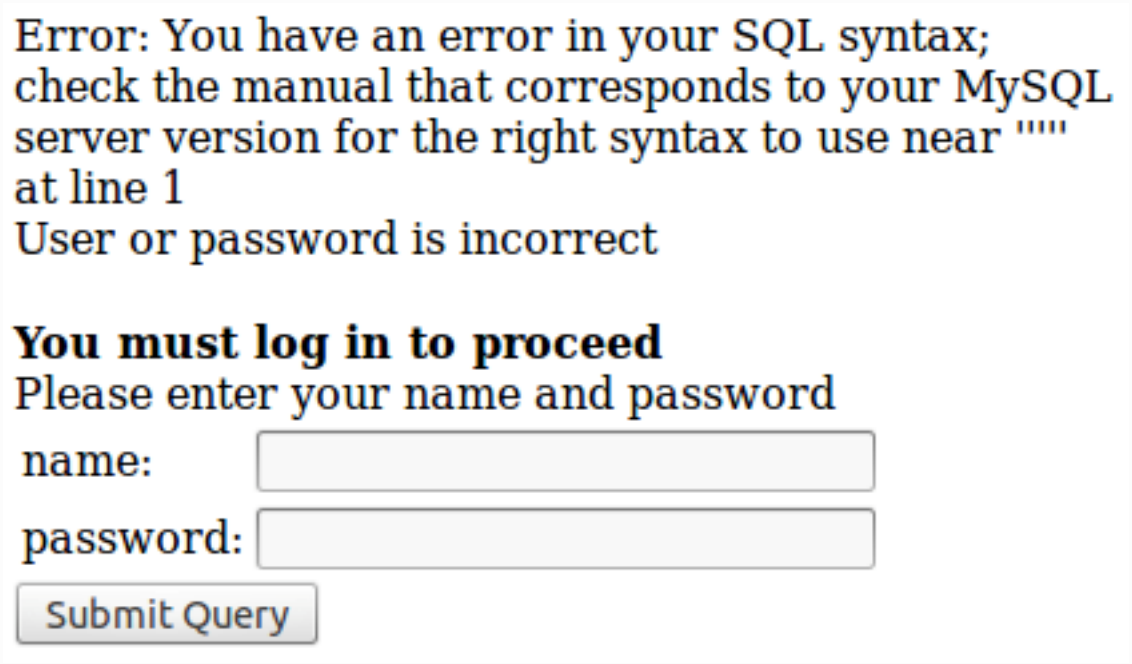
Form provides two values: login and password

What if password is a single quote? '

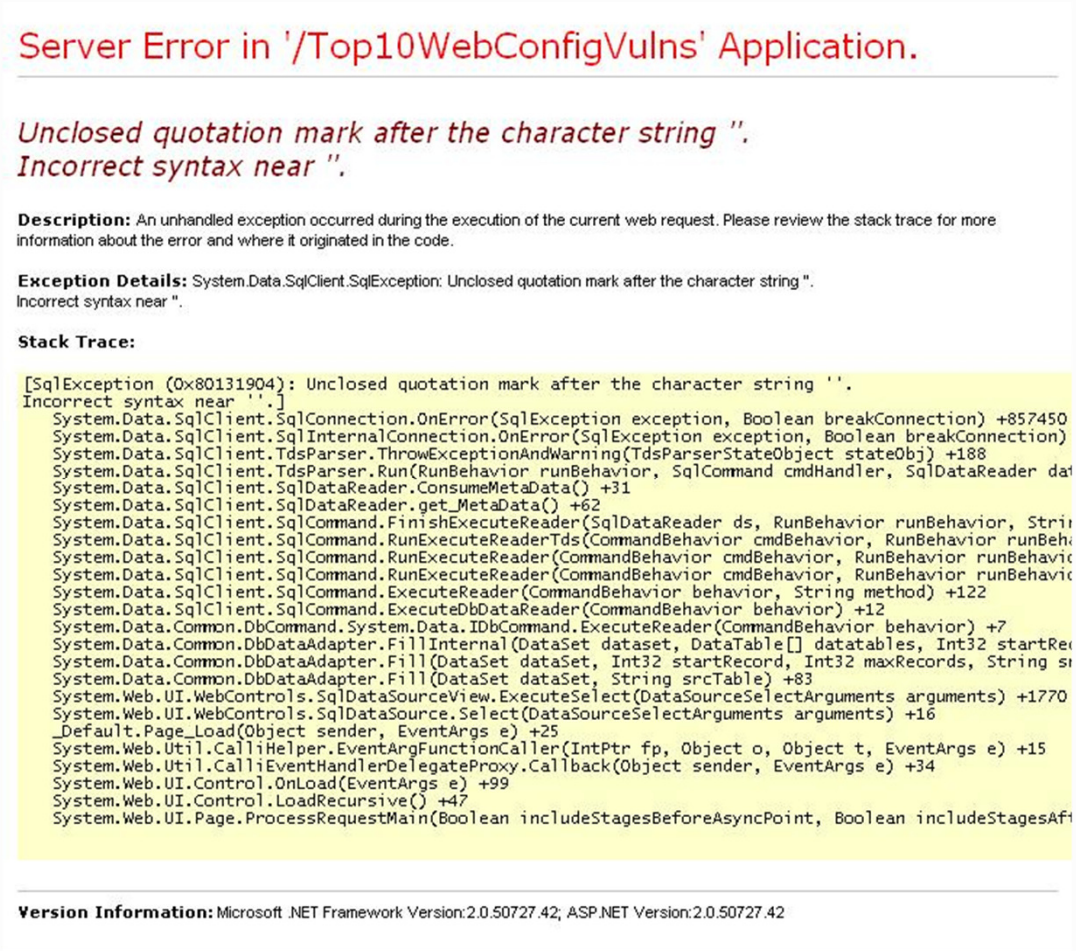
For login=admin and password=', query becomes:

```
SELECT user FROM users WHERE user='admin' AND password='''
```

SQL injection: detection



SQL Injection: Detection



Server Error in '/Top10WebConfigVulns' Application.

Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.

Stack Trace:

```
[SqlException (0x80131904): Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.]
   System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection) +857450
   System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection)
   System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj) +188
   System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader da
   System.Data.SqlClient.SqlDataReader.ConsumeMetaData() +31
   System.Data.SqlClient.SqlDataReader.get_MetaData() +62
   System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, Stri
   System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBeh
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavi
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavi
   System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method) +122
   System.Data.SqlClient.SqlCommand.ExecuteDbDataReader(CommandBehavior behavior) +12
   System.Data.Common.DbCommand.System.Data.IDbCommand.ExecuteReader(CommandBehavior behavior) +7
   System.Data.Common.DbDataAdapter.FillInternal(DataSet dataset, DataTable[] datatables, Int32 startRe
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, Int32 startRecord, Int32 maxRecords, String s
   System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, String srcTable) +83
   System.Web.UI.WebControls.SqlDataSourceView.ExecuteSelect(DataSourceSelectArguments arguments) +1770
   System.Web.UI.WebControls.SqlDataSource.Select(DataSourceSelectArguments arguments) +16
   _Default.Page_Load(Object sender, EventArgs e) +25
   System.Web.Util.CalliHelper.EventArgFunctionCaller(IntPtr fp, Object o, Object t, EventArgs e) +15
   System.Web.Util.CalliEventHandlerDelegateProxy.Callback(Object sender, EventArgs e) +34
   System.Web.UI.Control.OnLoad(EventArgs e) +99
   System.Web.UI.Control.LoadRecursive() +47
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAf
```

Version Information: Microsoft .NET Framework Version:2.0.50727.42; ASP.NET Version:2.0.50727.42

http://assets.devx.com/articlefigs/1705

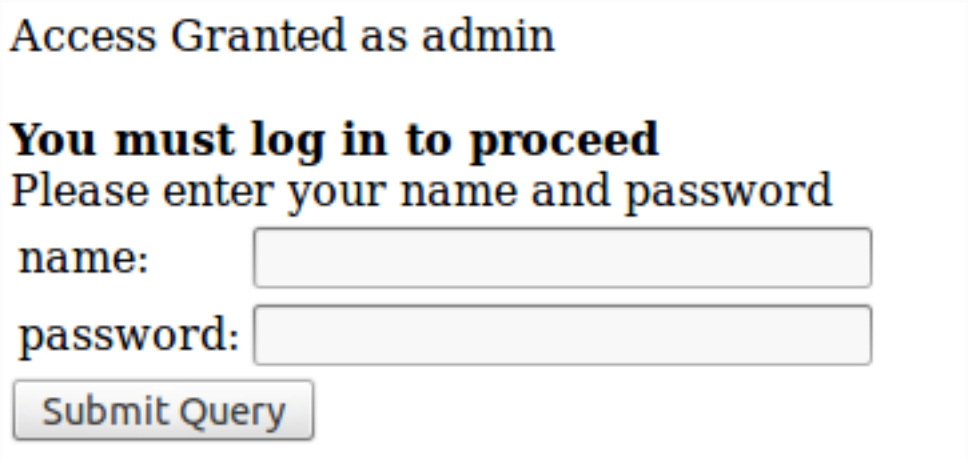
SQL injection: bypass simple password checking

- Form data is used to create an SQL statement
  - Without validation!
  - SQL code in form can be injected
- What if… password is ' or '1'='1

```
SELECT user FROM users WHERE user='admin'
       AND password='' or '1'='1'
```

- SQL statement is valid and always returns a row if the user exists

SQL injection:
bypass simple password checking



SQL Injection: bypass complex password checking

- SQL can store passwords in a ciphered format
  - Uses the PASSWORD function
  - Password stored in database cannot be obtained

- Typical validation query:

```
SELECT user FROM users WHERE user='$login'
    AND password=PASSWORD('$password')
```

- For `login=admin` and `password=') OR ('1'='1`, the query becomes:

```
SELECT user FROM users WHERE user='admin'
    AND password=PASSWORD('') OR ('1'='1')
```

Guess single password

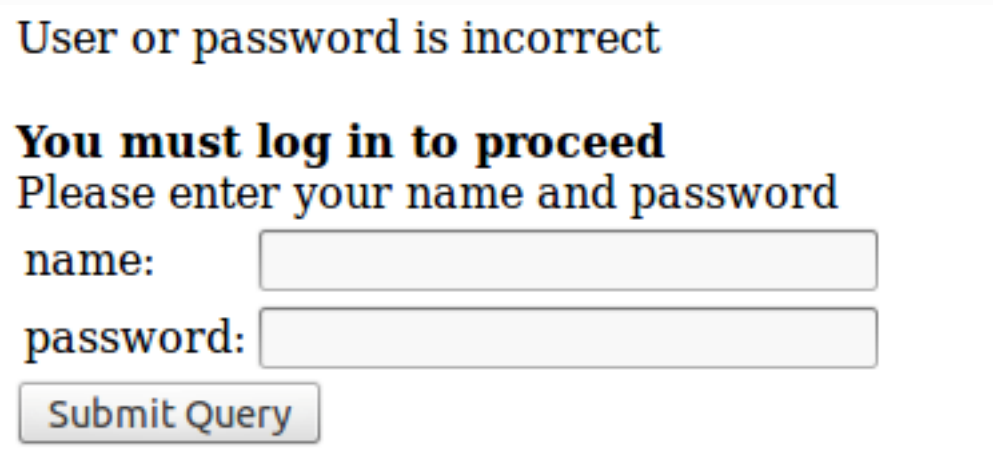
- More complex statement can be included in form fields
- Frequently, the only requirement is that they start and end with single quote (')
  - Because they will be inserted in attribute='injection'

- Does the password starts with 'a'?

```
' OR EXISTS(SELECT user FROM users
    WHERE user='admin' AND
    password LIKE 'a%') AND ''='
```

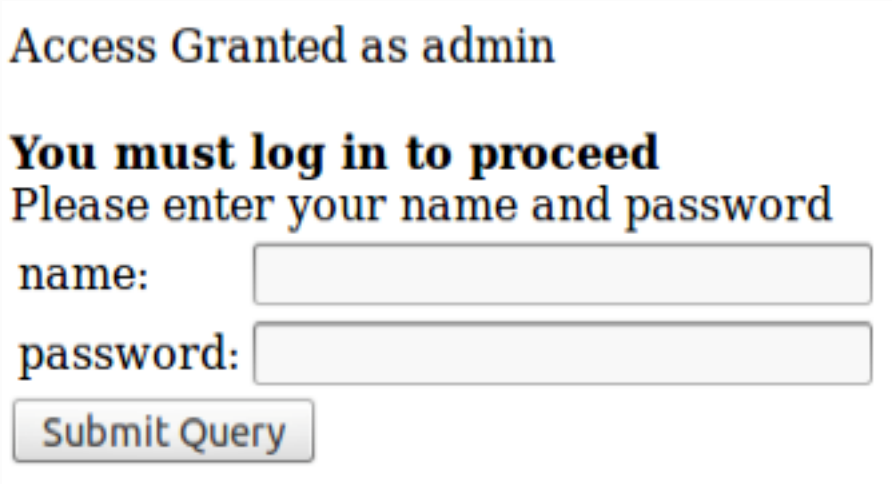
Guess simple password

```
SELECT user FROM users WHERE user='admin' AND
password='' OR EXISTS(SELECT user FROM users WHERE
user='admin' AND password LIKE 'a%') AND '' =''
```



Guess simple password

```
SELECT user FROM users WHERE user='admin` AND
password='' OR EXISTS(SELECT user FROM users WHERE
user='admin' AND password LIKE 'p%') AND '' =''
```



- Then we could try: **pa%** or **pa%a%**, etc..

Other possibilities

- Find table name:
  - Is there a users table in the current db?:

```
' OR EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_SCHEMA='test' AND TABLE_NAME='users') AND ''='
```

  - Is there any table starting by "p" in any db? :

```
' OR (SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'p%')>1 AND ''='
```

- Find database name
  - Starts by t?:

```
' OR EXISTS(SELECT 1 FROM users WHERE database() LIKE
't%') AND ''='
```

- Find columns, get columns by index, etc…

SQL injection: terminate query

- Two characters are particularly important
  - ; Terminates current query
    - Allows multiple queries in same request
  - -- terminates processing of all queries
    - Ignores syntax errors which may appear

Query 1

```
SELECT user FROM users WHERE user='admin' AND password='
```

`'; DROP TABLE user;`	`--'`
Query 2	**Ignored after --**

Mitigation: sanitize input data

- Sanitize form input data
  - Filter out dangerous characters
    - Username can only have letters
    - Passwords can only have letters and numbers
    - Emails must comply with RFC 2822
  - Escape dangerous characters
    - Avoid this
- Browser using Javascript
  - Can be bypassed doing direct queries or using tampering proxies
    - e.g. WebScarab
  - Automated tools can easily detect and bypass such methods
    - e.g. WebCruiser
- Server
  - Higher load in server
  - Much more effective!

Mitigation: sanitize input data

- Sanitizing input data based on quotes is insufficient!
  - If form is numeric, no quote is required.
  - e.g. PIN validation

```
SELECT user FROM users WHERE
   user='admin' AND pin=12 or 1=1
```

- Validation must take in consideration actual data
  - Sanitize as much as possible

Mitigation: sanitize input data

- Escaping doesn't really help in all cases
  - e.g. typical escape is ' → ''
- Providing ' OR '1'='1 results in:

```
SELECT user FROM users WHERE user='admin' AND
password='' OR ''1''='1'
```

- The resulting query is invalid, no harm done
- What about \'; DROP TABLE users; --
  - \' is expanded to \'', '\'' is a valid string with just one character (the single quote); the table is dropped!
- MySQL provides own sanitization methods: mysql_real_escape_string()

Mitigation: prepared queries

- Instead of building query string, let SQL libraries compile the query.
  - Separation between Query and Parameters

- Three steps required:
  - Preparation
  - Bind parameters
  - Execution

Mitigation: prepared queries

- Query Preparation:

```
$s = mysql->prepare("SELECT user FROM
users WHERE user=? AND pin=?")
```

- Parameter binding:

```
$s->bind_param("s",$login);
$s->bind_param("i",$password);
```

- Query execution:

```
$s->execute();
```

Mitigation: other methods

- Limit data permissions according to user needs
  - Do not grant DROP, or Write methods for read-only application
- Use stored procedures
- Configure error reporting appropriately
  - Detailed error reporting for developers
  - Limited error reporting for users
- Isolate servers to reduce compromise of neighbor hosts