

'-- sql injection

the old dog of data security



'-- the evergreen vulnerability

And although declining in recent years, it just won't go away.

- 6.7% of all vulnerabilities in open-source.
- 10% of all vulnerabilities in closed-source.

(2024)



'-- some greatest hits

You may have heard about some of these fine companies?

- Sony Playstation Network
- Equifax
- Yahoo
- Epic Games / Fortnite
- SQL Server Reporting Services
- 7-Eleven
- Sony Pictures
- Marriott International
- TSA



The good news is, it's relatively easy to fix.



learning objectives

- What is a SQL injection
- What are the risks
- What to look for
- How to fix your code

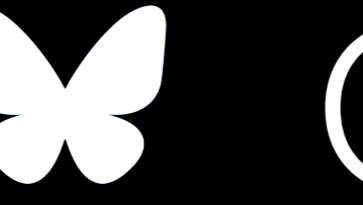




so who am I?

'-- daniel hutmacher

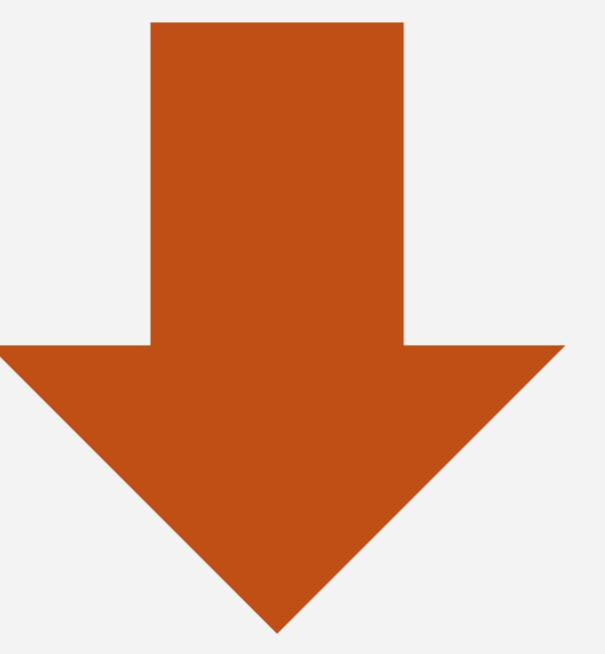
sql server consultant
conference organizer
dog person

sqlsunday.com
daniel@strd.co
 [@dhma.ch](https://twitter.com/dhma.ch)
 [/in/danielhutmacher/](https://www.linkedin.com/in/danielhutmacher/)



what is sql injection?

```
powershell  
$email = "daniel@strd.co"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



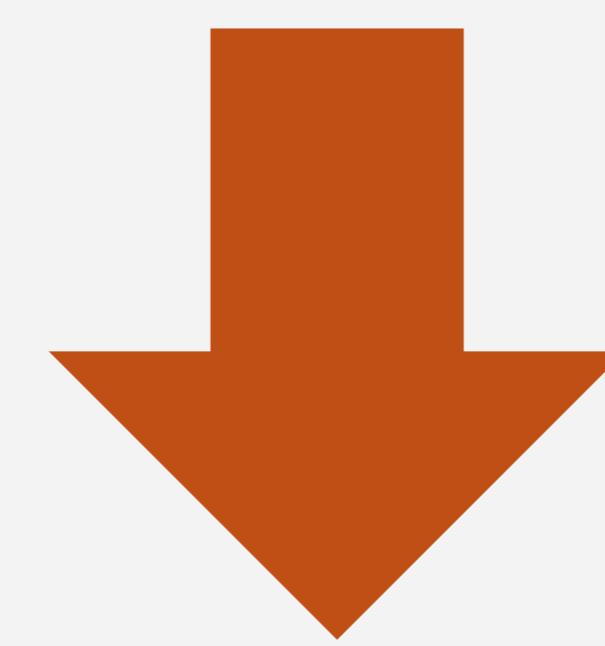
```
sql  
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co';
```



what is sql injection?

powershell

```
$email = "daniel@strd.co' OR 1=1 --"  
  
$sqlQuery = @'  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co' OR 1=1 --';
```



types of sql injection

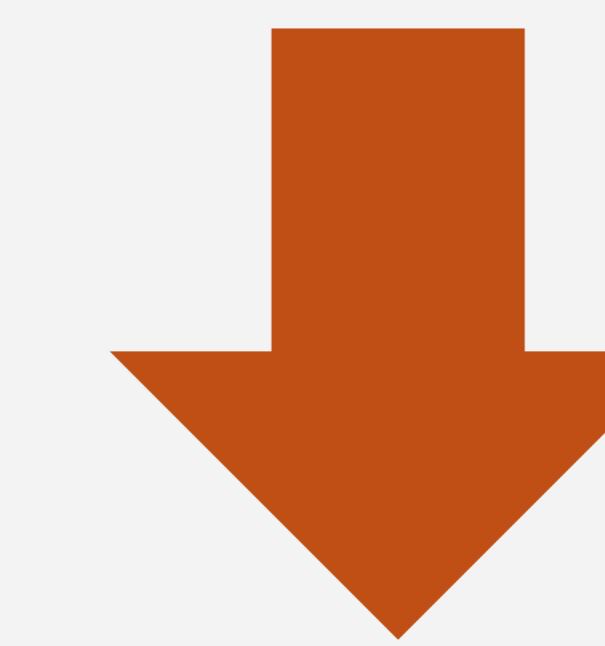


types of sql injection

out-of-band attack

powershell

```
$email = "daniel@strd.co'; CREATE LOGIN legit WITH  
PASSWORD='1234'; GRANT CONTROL SERVER TO legit; --"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co'; CREATE LOGIN legit WITH  
PASSWORD='1234'; GRANT CONTROL SERVER TO legit; --';
```

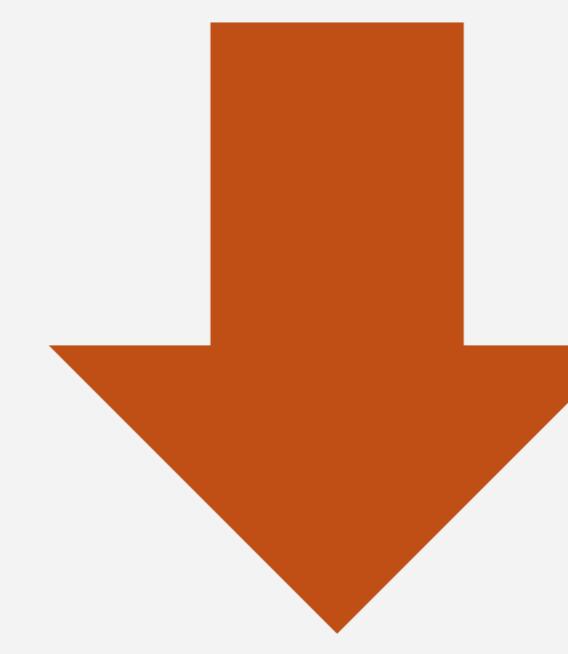


types of sql injection

out-of-band attack

powershell

```
$email = "daniel@strd.co'; xp_cmdshell 'del  
C:\Windows\System32\*'; --"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co'; xp_cmdshell 'del  
C:\Windows\System32\*'; --';
```

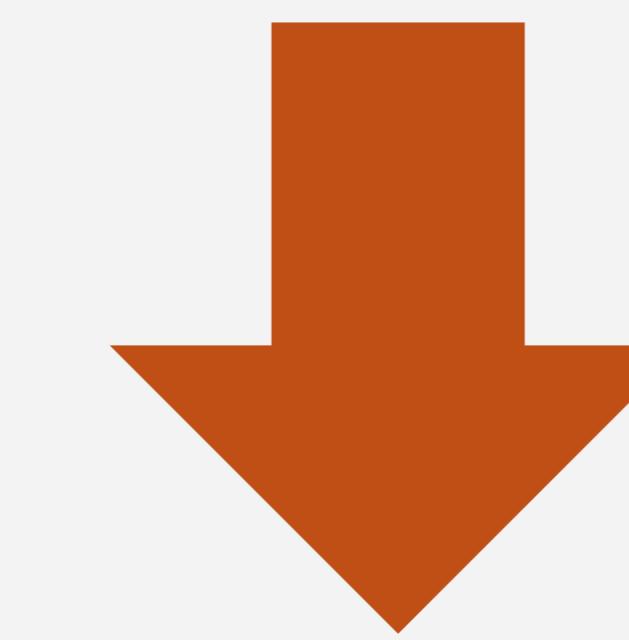


types of sql injection

error-based injection

powershell

```
$email = "daniel@strd.co' OR 1=CAST(@@SERVERNAME AS int) --"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co' OR 1=CAST(@@SERVERNAME AS int) --';
```

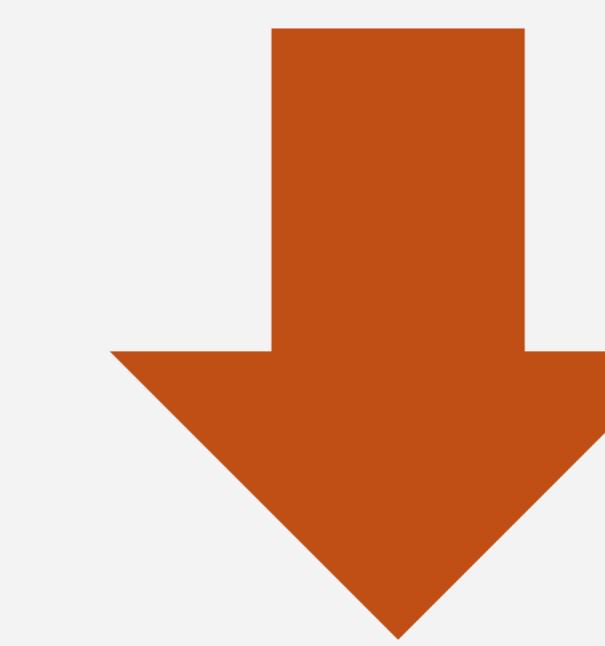


types of sql injection

error-based injection

powershell

```
$email = "daniel@strd.co' OR 1=CAST(@@SERVERNAME AS int) --"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co' OR 1=CAST(@@SERVERNAME AS int) --';
```

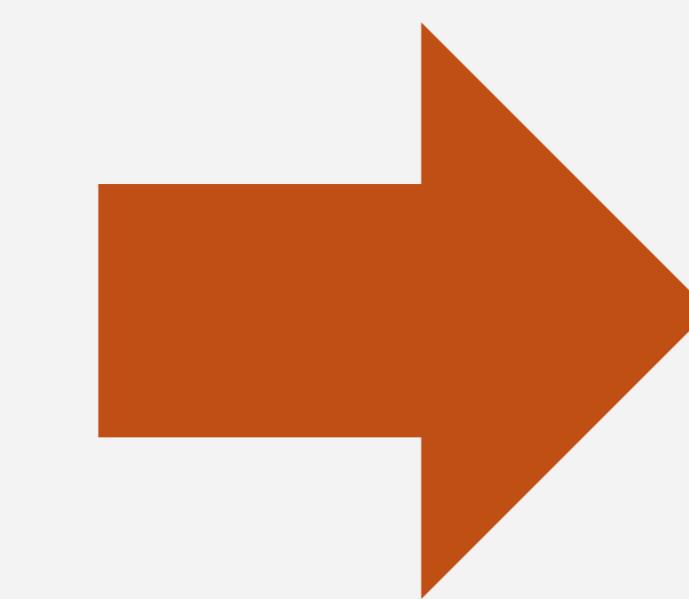


Msg 245, Level 16, State 1, Line 1
Conversion failed when converting the nvarchar value 'SQL01' to
data type int.

types of sql injection

error-based injection

-- demo



Test-Injection.ps1

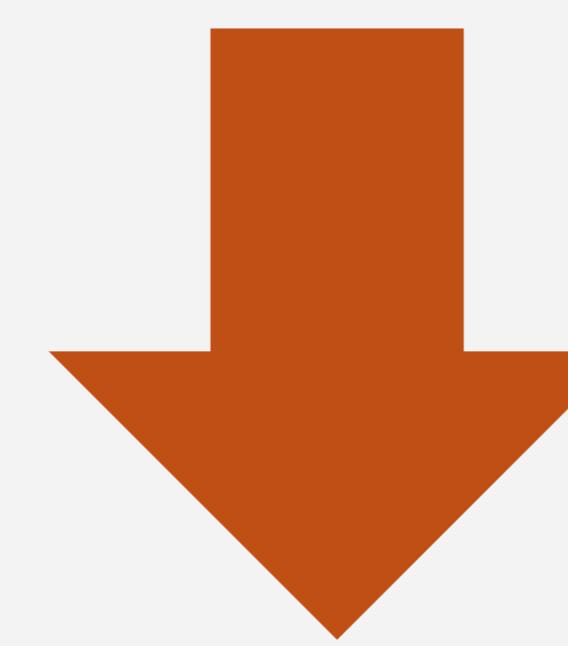


types of sql injection

blind injection

powershell

```
$email = "daniel@strd.co'; IF (SUSER_NAME()='sa') WAITFOR DELAY  
'00:00:01'; --"  
  
$sqlQuery = @"  
    SELECT id, firstName, lastName  
    FROM dbo.Users  
    WHERE email='$email';  
"@
```



sql

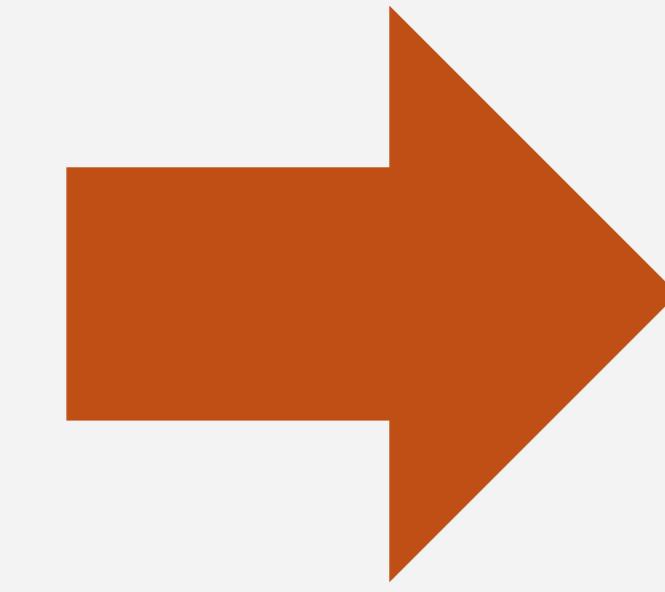
```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co'; IF (SUSER_NAME()='sa') WAITFOR  
DELAY '00:00:01'; --';
```



types of sql injection

blind injection

sqlmap demo



```
Daniels zsh
~/Desktop/sqlmap ↵ python3 sqlmap.py -u http://localhost:3000/login --data="username=hello&password=helloagain" --tables
--columns -dbms=MSSQL -D InjectionDemo
```

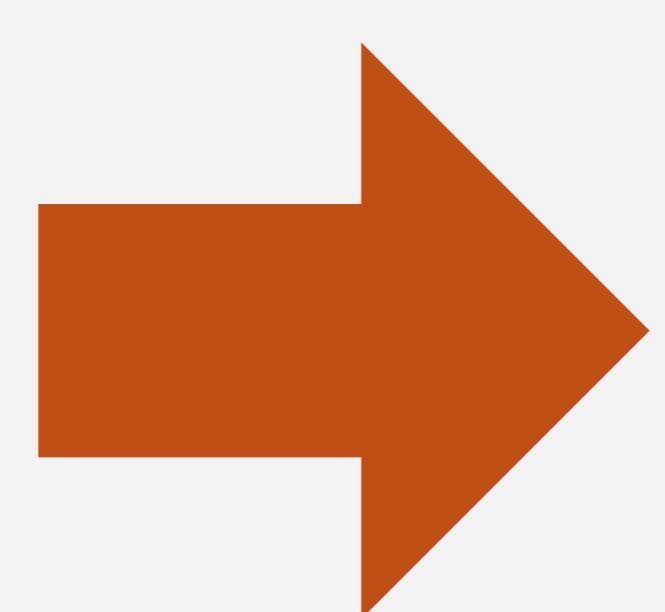


let's inject some sales

because we're worth it.



-- demo



Injection demo code.txt

dynamic sql

ETL frameworks can be vulnerable, too.

ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	Updated
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated



dynamic sql

ETL frameworks can be vulnerable, too.



ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	Updated
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated

sql

```
SELECT TOP (1) @sql='
    INSERT INTO '+Dest_object+
    ' SELECT * FROM '+Source_object+
    ' WHERE '+TimestampColumn+ '>' +
    '(SELECT MAX('+TimestampColumn+') FROM '+Dest_object+'));
FROM Metadata.ETL_flows;

EXEC(@sql);
```

dynamic sql

ETL frameworks can be vulnerable, too.

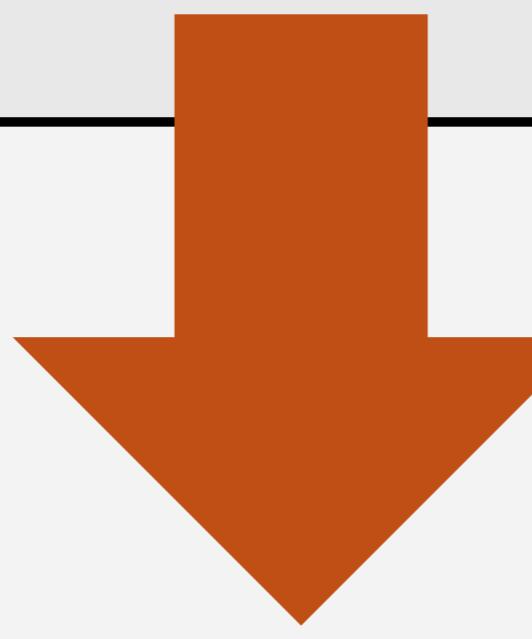


ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	Updated
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated

sql

```
SELECT TOP (1) @sql='
    INSERT INTO '+Dest_object+
    ' SELECT * FROM '+Source_object+
    ' WHERE '+TimestampColumn+ '>' +
    '(SELECT MAX('+TimestampColumn+') FROM '+Dest_object+'));
FROM Metadata.ETL_flows;

EXEC(@sql);
```



sql

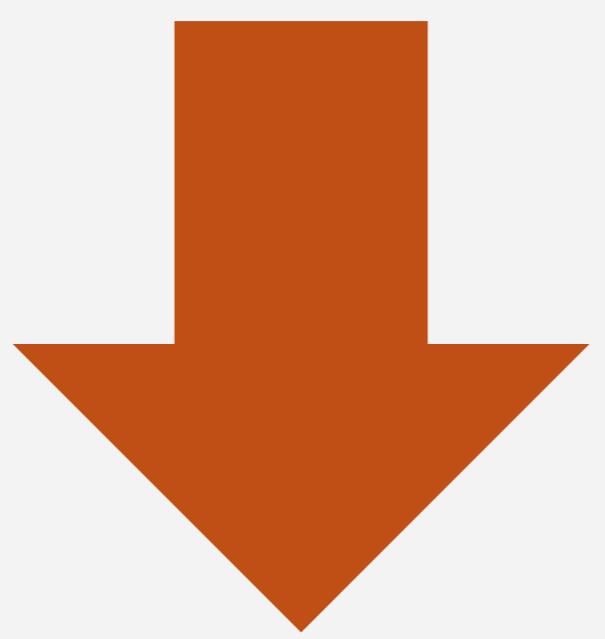
```
INSERT INTO dbo.Currencies
SELECT * FROM Staging.Currencies
WHERE Updated>(SELECT MAX(Updated) FROM dbo.Currencies);
```

dynamic sql

ETL frameworks can be vulnerable, too.



ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'--
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated



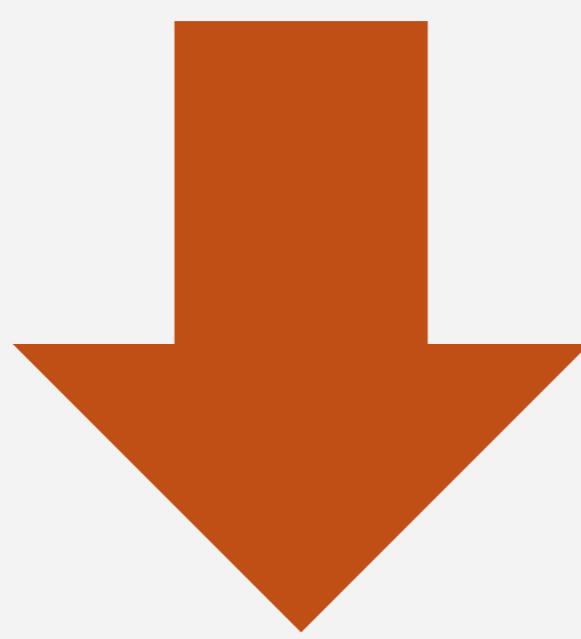
```
sql
INSERT INTO dbo.Currencies
SELECT * FROM Staging.Currencies
WHERE 1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET
TimestampColumn='Updated'-->(SELECT MAX(1=0; DROP TABLE
dbo.AuditLog; UPDATE Metadata.ETL_flows SET
TimestampColumn='Updated'--
) FROM dbo.Currencies);
```

dynamic sql

ETL frameworks can be vulnerable, too.



ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'--
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated



1. finish the INSERT:

sql

```
INSERT INTO dbo.Currencies
SELECT * FROM Staging.Currencies
WHERE 1=0;

DROP TABLE dbo.AuditLog;

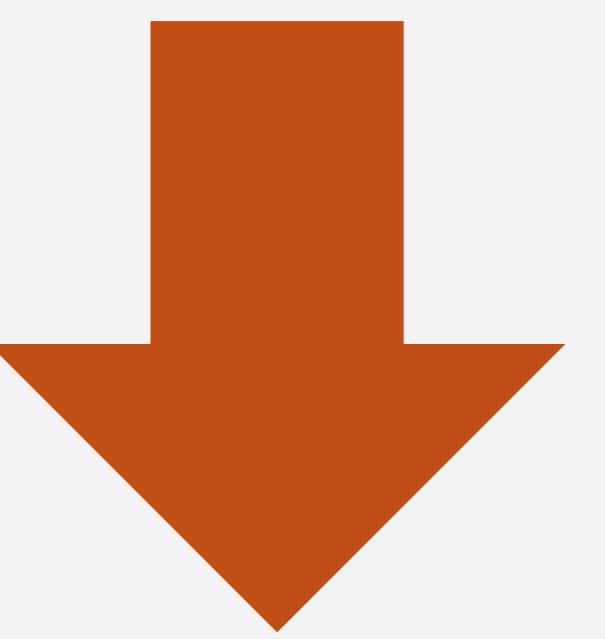
UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'-->(SELECT
MAX(1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET
TimestampColumn='Updated'--
) FROM dbo.Currencies);
```

dynamic sql

ETL frameworks can be vulnerable, too.



ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'--
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated



1. finish the INSERT:

```
INSERT INTO dbo.Currencies  
SELECT * FROM Staging.Currencies  
WHERE 1=0;
```

2. do bad things:

sql

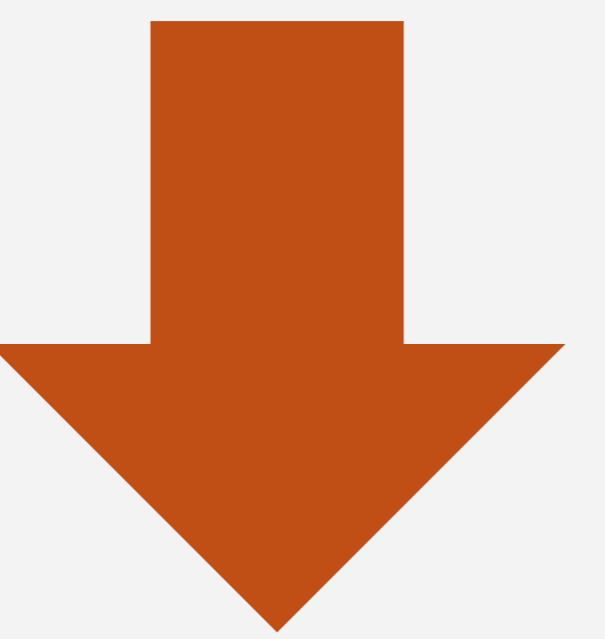
```
DROP TABLE dbo.AuditLog;  
  
UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'-->(SELECT  
MAX(1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET  
TimestampColumn='Updated'--  
) FROM dbo.Currencies);
```

dynamic sql

ETL frameworks can be vulnerable, too.



ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'--
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated



1. finish the INSERT:
2. do bad things:
3. remove evidence:

sql

```
INSERT INTO dbo.Currencies
SELECT * FROM Staging.Currencies
WHERE 1=0;

DROP TABLE dbo.AuditLog;

UPDATE Metadata.ETL_flows SET TimestampColumn='Updated'-->(SELECT
MAX(1=0; DROP TABLE dbo.AuditLog; UPDATE Metadata.ETL_flows SET
TimestampColumn='Updated'--
) FROM dbo.Currencies);
```



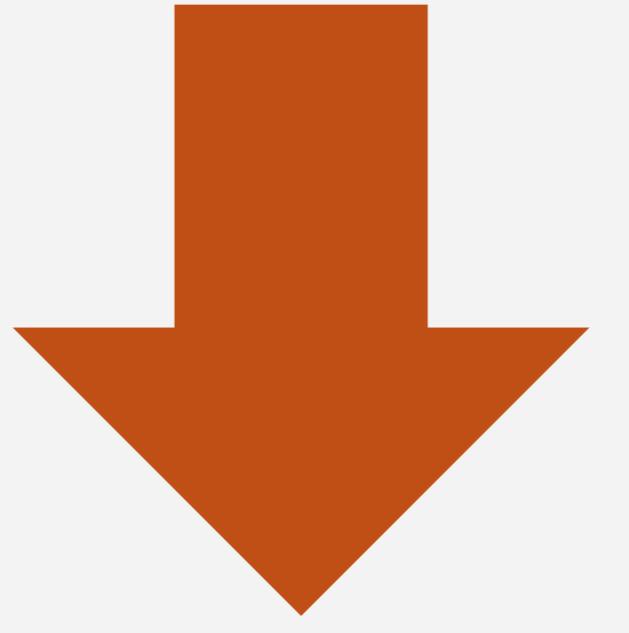
but wait, there's more:

connection string injection

connection string injection



```
powershell  
  
$username = "daniel"  
$password = "Pa$$w0rd!"  
  
$connectionString = "Server=sql01.strd.co;Database=Prod_DB;User  
Id=$username;Password=$password;"
```

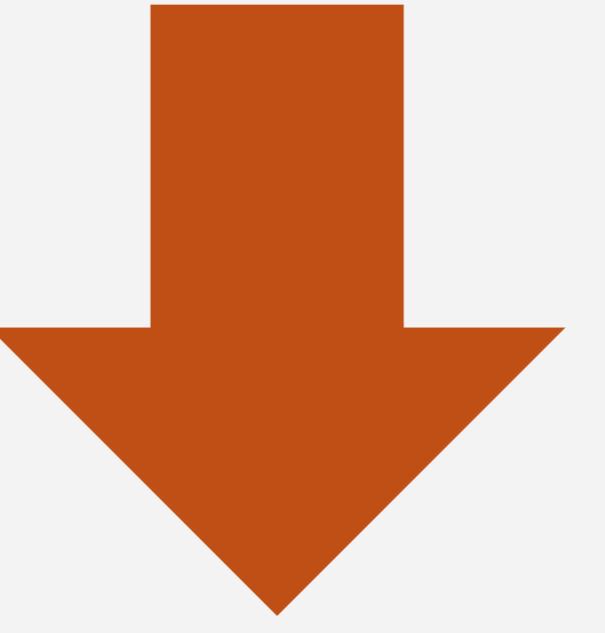


```
connection string  
  
Server=sql01.strd.co;Database=Prod_DB;User  
Id=daniel;Password=Pa$$w0rd!;
```

connection string injection



```
powershell  
  
$username = "daniel"  
$password = "Pa$$w0rd!;Integrated Security=True"  
  
$connectionString = "Server=sql01.strd.co;Database=Prod_DB;User  
Id=$username;Password=$password;"
```



```
connection string  
  
Server=sql01.strd.co;Database=Prod_DB;User  
Id=daniel;Password=Pa$$w0rd!;Integrated Security=True;
```

'-- mitigations

- Parameterizing inputs
- Manually sanitizing inputs
- Using an ORM
- Using stored procedures
- Restrictive permissions
- Disabling error messages on web sites
- Web application firewalls (WAF)
- Rate-limit your web app or API





mitigations

parameterizing inputs

```
c#  
string query = "SELECT id, firstName, lastName FROM dbo.Users  
WHERE email = @Email";  
  
Command command = new SqlCommand(query, connection)  
  
command.Parameters.Add(  
    new SqlParameter("@Email", SqlDbType.NVarChar) {  
        Value = email  
    }  
);
```





mitigations

parameterizing dynamic t-sql

```
SET @sql=N'  
UPDATE dbo.Users  
SET firstName=@first, lastName=@last  
WHERE email=@email;'  
  
EXECUTE sys.sp_executesql  
@sql,  
@params = N'@first nvarchar(100), @last nvarchar(100), @email varchar(255)',  
@first = @new_firstname,  
@last = @new_lastname,  
@email = @email;
```





mitigations

use an ORM

using parameterized SQL
(EF Core 2.0+)

c#

```
var email = "daniel@strd.co";  
  
var users = context.Users  
    .FromSqlInterpolated($"SELECT id, firstName, lastName FROM  
dbo.Users WHERE email = {email}")  
    .ToList();
```

using DbSet.SqlQuery
(EF6+)

c#

```
var email = "daniel@strd.co";  
  
var users = context.Users  
    .SqlQuery("SELECT id, firstName, lastName FROM dbo.Users  
WHERE email = @p0", email)  
    .ToList();
```





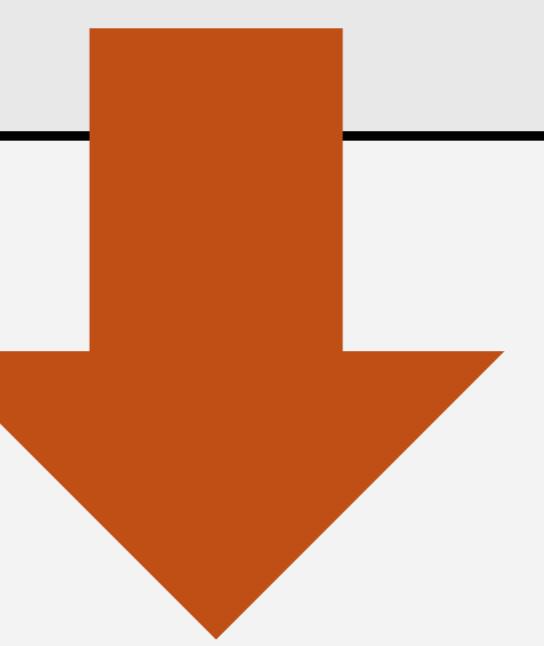
mitigations

sanitizing dynamic sql

ID	Source_object	Dest_object	Dependency	TimestampColumn
1	Staging.Currencies	dbo.Currencies	NULL	Updated
2	Staging.CurrencyRates	dbo.Fx_Rates	1	Updated
3	Staging.Contracts	dbo.Contracts	2	Updated

```
sql| SELECT TOP (1) @sql='
    INSERT INTO '+Dest_object+
    'SELECT * FROM '+Source_object+
    ' WHERE '+QUOTENAME(TimestampColumn)+ '>'+
    '(SELECT MAX('+QUOTENAME(TimestampColumn)+')) FROM
    '+Dest_object+';
    FROM Metadata.ETL_flows;

    EXEC(@sql);'
```



```
sql| INSERT INTO dbo.Currencies
    SELECT * FROM Staging.Currencies
    WHERE [Updated]>(SELECT MAX([Updated]) FROM dbo.Currencies);
```



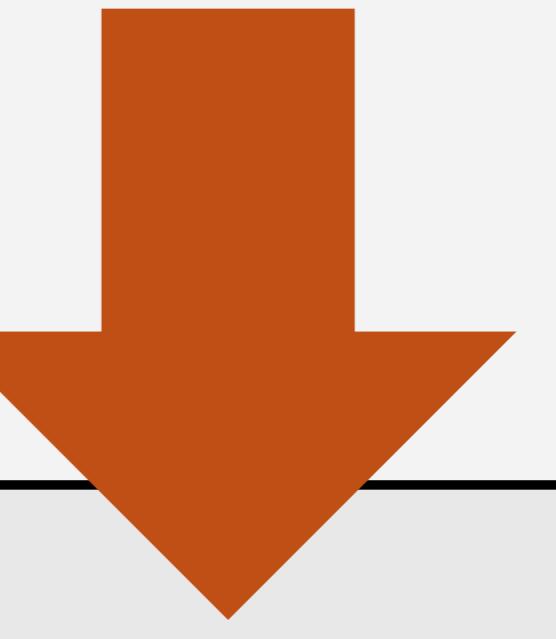
mitigations

sanitizing dynamic sql

I ❤️ QUOTENAME()

sql

```
SELECT QUOTENAME('hello');  
SELECT QUOTENAME('object', '[');  
SELECT QUOTENAME('quote', '\"');  
SELECT QUOTENAME('apostrophe', ''');
```



```
[hello]  
[object]  
"quote"  
'apostrophe'
```



mitigations

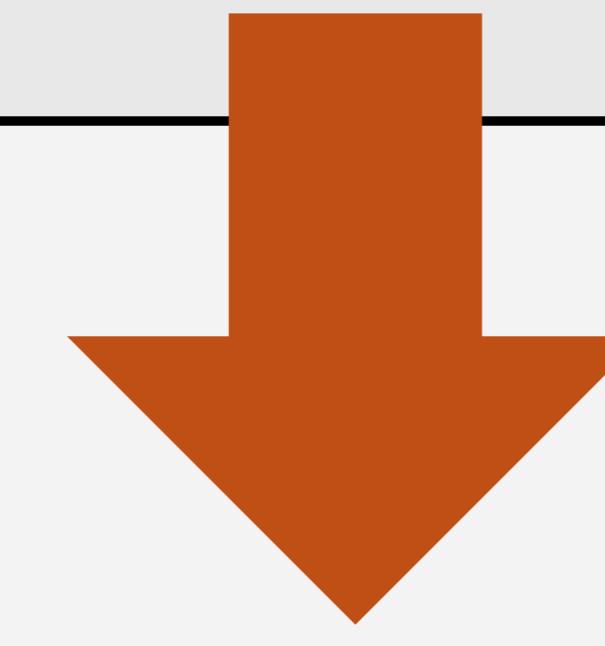
sanitizing dynamic sql

- OBJECT_ID() returns the unique id of a table/view/etc.
- OBJECT_NAME() returns the name of an object.
- OBJECT_SCHEMA_NAME returns the name of the schema of an object.
- QUOTENAME quotes (and sanitizes) a string.

sql

```
-- DECLARE @Source_object nvarchar(100)='Staging.Currencies';

SELECT
    QUOTENAME(OBJECT_SCHEMA_NAME(OBJECT_ID(@Source_object)))+
    N'.'+
    QUOTENAME(OBJECT_NAME(OBJECT_ID(@Source_object))');
```



[Staging].[Currencies]



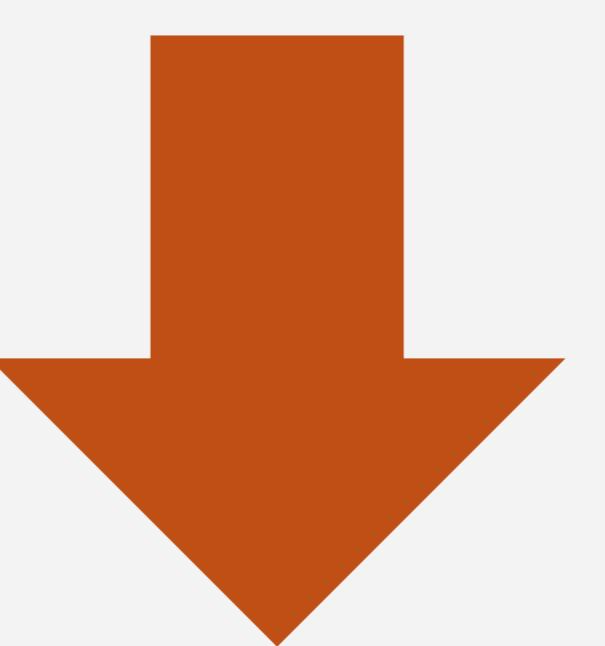


mitigations

sanitizing inputs

powershell

```
$email = "daniel@strd.co'; IF (SUSER_NAME()='sa') WAITFOR DELAY  
'00:00:01'; --"  
  
$email_escaped = $email -replace "'", ""  
  
$sqlQuery = @"  
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='$email_escaped';  
"@
```



sql

```
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE email='daniel@strd.co''; IF (SUSER_NAME()='sa') WAITFOR  
DELAY '00:00:01'; --';
```



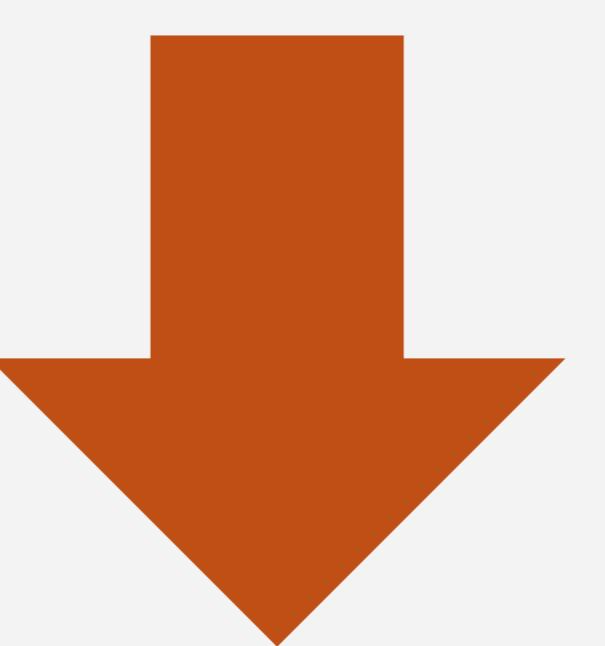


mitigations

sanitizing inputs

powershell

```
$id = "123; DROP TABLE dbo.Students--"  
  
$id_escaped = [int]$id  
  
$sqlQuery = @'  
SELECT id, firstName, lastName  
FROM dbo.Users  
WHERE id=$id_escaped;  
"@
```



```
InvalidArgumentException: Cannot convert value "123; DROP TABLE  
dbo.Students--" to type "System.Int32". Error: "The input string  
'123; DROP TABLE dbo.Students--' was not in a correct format."
```



mitigations

sanitizing inputs

- There are theorized highly advanced exploits, where you could use specially crafted unicode strings with multi-byte characters.
- This is the only realistic option for languages where no parameterization can be done, like PowerShell.





mitigations

stored procedures with
restrictive permissions

- Only grant the application account EXECUTE permissions on stored procedures
- Do not grant SELECT, UPDATE, INSERT or DELETE

✋ This is not a sufficient mitigation on its own, but a good defense-in-depth strategy when combined with other mitigations.





mitigations

stored procedures with restrictive permissions

- If you're brave enough, actually denying access to the system DMVs would prevent enumeration of tables, views, etc, but this could affect your application.

👉 This is not a sufficient mitigation on its own, but a good defense-in-depth strategy when combined with other mitigations.

sql

```
DENY SELECT ON SCHEMA::sys TO [databaseUser];  
DENY SELECT ON SCHEMA::INFORMATION_SCHEMA TO [databaseUser];
```





mitigations

disabling web server error messages

- Not showing error messages on your web page makes it much harder for attackers to use error-based injections

👉 This is not a sufficient mitigation on its own, but a good defense-in-depth strategy when combined with other mitigations.





mitigations

web application firewalls

- A web application firewall (WAF) can detect sql injection patterns and block the client from running repetitive exploratory queries.

👉 This is not a sufficient mitigation on its own, but a good defense-in-depth strategy when combined with other mitigations.





mitigations

rate limiting

- Timing-based attacks require *a lot* of http calls. Rate limiting your site may mitigate timing-based attacks, but is not a fully secure solution on its own.

👉 This is not a sufficient mitigation on its own, but a good defense-in-depth strategy when combined with other mitigations.





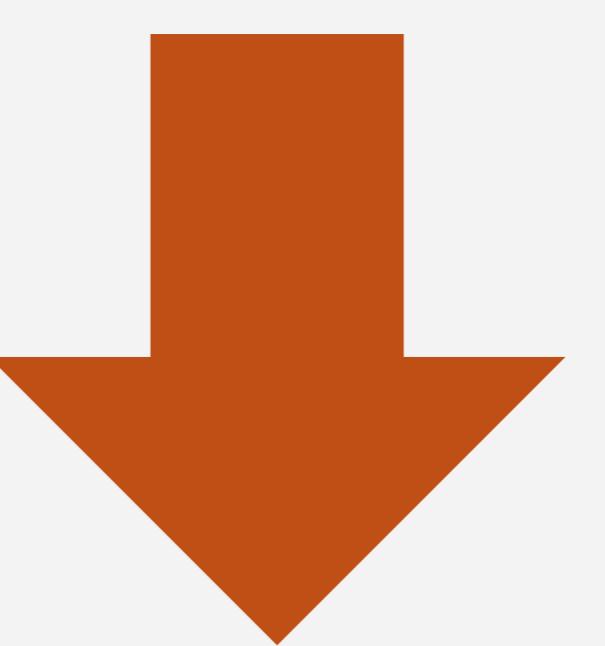
mitigations

sanitize connection strings

- Quote strings with apostrophes or double quotes
- Escape apostrophes/quotes accordingly

Not a very common vector, but relatively easy to mitigate.

```
Server=sql01.strd.co;Database=Prod_DB;User  
Id=daniel;Password=Pa$$w0rd!;Integrated Security=True;
```



connection string

```
Server="sql01.strd.co";Database="Prod_DB";User  
Id="daniel";Password="Pa$$w0rd!;Integrated Security=True";
```



'-- questions?



'-- thank you



contact

sqlsunday.com

daniel@strd.co

 [@dhma.ch](https://twitter.com/dhma)

 [/in/danielhutmacher/](https://www.linkedin.com/in/danielhutmacher/)

slides and demos

github.com/sqlsunday/presentations

github.com/sqlsunday/injection-demo