

Simulation Report (Individual Version)

Course: SQL Server Development

Database: AdventureWorks2019

Student Name / ID: Yung-Lun Lee

1. Introduction

In this simulation, I developed and tested Dynamic SQL in both secure and vulnerable ways. I created:

1. A secure procedure using parameterized SQL
2. A vulnerable procedure using string concatenation

I also added input validation, error handling, and execution logging.

The goal was to compare how these two approaches behave in terms of security and performance.

2. System Design

2.1 ExecutionLog Table

I used the existing `Reporting.ExecutionLog` table to store:

- Procedure name
- Executed SQL
- Execution time
- Error message
- User and host
- Duration in milliseconds
- Parameter values

These logs help me review activity, compare results, and identify errors.

2.2 Secure Procedure

The secure procedure:

- Uses `sp_executesql` with parameters
- Prevents SQL injection
- Has TRY/CATCH error handling

- Logs every execution
- Rejects suspicious input (Task 4)

```

--Task 2 - Secure DS Implementation
CREATE OR ALTER PROCEDURE Reporting_DynamicSalesSummarySecure
(
    @Territory NVARCHAR(50) = NULL,
    @SalesPerson NVARCHAR(100) = NULL,
    @Category NVARCHAR(50) = NULL,
    @StartDate DATE = NULL,
    @EndDate DATE = NULL
)
AS
BEGIN
    SET NOCOUNT ON;
    -----
    -- Variable Declarations
    -----
    DECLARE @SQL NVARCHAR(MAX) = N'';
    DECLARE @ParamDef NVARCHAR(MAX);
    DECLARE @EndDateTime DATETIME2(3) = SYSUTCDATETIME();
    DECLARE @EndTime DATETIME2(3);
    DECLARE @DurationMS INT;
    DECLARE @ParameterValues NVARCHAR(MAX) = N'';

    BEGIN TRY
        -----
        -- Basic Input Validation (Full version in Task 4)
        -----
        IF @Territory LIKE '%;%' OR
            @SalesPerson LIKE '%;%' OR
            @Category LIKE '%;%'
        BEGIN
            ...
        END
    
```

2.3 Vulnerable Procedure

The vulnerable version:

- Uses raw string concatenation
- Runs the SQL through `EXEC(@SQL)`
- Has no validation
- Allows SQL injection on purpose for demonstration

The screenshot shows two separate sessions in SQL Server Management Studio (SSMS) against the AdventureWorks2019 database.

Session 1:

```

329    ON [AdventureworksLT].[dbo].[SalesTerritory]
330    @HOST_NAME,
331    @DurationMS,
332    @ParameterValues
333    );
334
335    THROW;
336  END CATCH
337  GO
338
339  --test
340  EXEC Reporting.DynamicSalesSummaryVulnerable
341  @Territory = '*** OR 1=1 ***';
342
343
344
345
346
347
348
349
350
351
352

```

Session 2:

```

324    THREW;
325  END CATCH
326  GO
327
328  --test
329  EXEC Reporting.DynamicSalesSummaryVulnerable
330  @Territory = '*** OR 1=1 ***';
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352

```

Results for Session 1:

BusinessEntityID	Name	Territory	Category	OrderDate	TotalDue	
1	282	David Campbell	Northwest	Accessories	2011-05-31 00:00:00.000	16158.690
2	282	Joel Sativa	Canada	Accessories	2011-05-31 00:00:00.000	40487.723
3	275	Michael Blythe	Northeast	Accessories	2011-05-31 00:00:00.000	4216.058
4	278	Garrett Vargas	Canada	Accessories	2011-05-31 00:00:00.000	8773.681
5	281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	11036.394
6	281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	12832.909
7	279	Tsvi Reiter	Southeast	Accessories	2011-05-31 00:00:00.000	15524.068
8	283	David Campbell	Northwest	Accessories	2011-05-31 00:00:00.000	48204.066
9	281	Shu Ito	Southwest	Accessories	2011-05-31 00:00:00.000	43662.496
10	275	Michael Blythe	Southeast	Accessories	2011-05-31 00:00:00.000	23126.45
11	279	Tsvi Reiter	Southeast	Accessories	2011-05-31 00:00:00.000	23042.186
12	274	Stephan Tang	Northwest	Accessories	2011-05-31 00:00:00.000	7688.297
13	275	Michael Blythe	Northwest	Accessories	2011-05-31 00:00:00.000	7688.297
14	282	Joel Sativa	Canada	Accessories	2011-07-01 00:00:00.000	590.0178
15	279	Tsvi Reiter	Southwest	Accessories	2011-07-01 00:00:00.000	26146.807
16	280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	36806.480
17	281	Shu Ito	Central	Accessories	2011-07-01 00:00:00.000	10580.080
18	276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	13042.679
19	276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	46665.951
20	279	Tsvi Reiter	Southeast	Accessories	2011-07-01 00:00:00.000	157343.7...
21	280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	23223.597
22	282	Joel Sativa	Canada	Accessories	2011-07-01 00:00:00.000	24459.114
23	277	Jillian Carson	Central	Accessories	2011-07-01 00:00:00.000	130416.4...
24	280	Pamela Ann...	Northwest	Accessories	2011-07-01 00:00:00.000	36556.886
25	276	Linda Mitchell	Southwest	Accessories	2011-07-01 00:00:00.000	57683.232
26	278	Garrett Vargas	Canada	Accessories	2011-07-01 00:00:00.000	57391.428

Session 2:

```

324    THREW;
325  END CATCH
326  GO
327
328  --test
329  EXEC Reporting.DynamicSalesSummaryVulnerable
330  @Territory = '*** OR 1=1 ***';
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352

```

Results for Session 2:

LogID	ProcedureName	ExecutedSQL	ExecutionDate	ErrorMessage	ExecutedBy	HostName	DurationMS	ParameterValues
1	Reporting.DynamicSalesSummaryVulnerable	SELECT sp...[REDACTED]	2025-11-16 20:02:50.977	NULL	ANGELALEECCAB\junjun	ANGELALEECCAB	641	Territory='*** OR 1=1 ***'; SalesPerson=NULL; Category=...
2	Reporting.SafeUpdateProductCost	UPDATE Production.Product...	2025-11-10 15:53:20.330	Product not found	NULL	NULL	NULL	NULL

2.4 Input Validation

I added simple validation to the secure procedure.
If a parameter includes keywords such as:

- `--`
- `DROP`
- `INSERT`
- `EXEC`

The procedure immediately stops, logs the attempt as `RejectedInput`, and returns a safe error message.

The screenshot displays the SQL Server Management Studio (SSMS) interface with two separate windows, each showing a query editor and its results.

Top Window:

- Object Explorer:** Shows the database structure for AdventureWorks2019, including the BookstoreDB database.
- Query Editor:** Contains the following T-SQL code:

```
565     SYSUTCDATETIME(),  
566     ERROR_MESSAGE(),  
567     ORIGINAL_LOGIN(),  
568     HOST_NAME(),  
569     @DurationMS,  
570     @ParameterValues  
571 );  
572  
573     THROW;  
574 END CATCH  
575  
576 GO  
577  
--test  
578 EXEC Reporting.DynamicSalesSummarySecure  
579     @Territory = 'Northwest'; DROP TABLE Sales.SalesOrderHeader ---;
```
- Messages:** Displays the error message:

```
Msg 51001, Level 16, State 1, Procedure Reporting.DynamicSalesSummarySecure, Line 70 [Batch Start Line 578]  
Input rejected due to security policy.
```
- Completion Time:** 2025-11-16T15:08:25.1152718-05:00

Bottom Window:

- Object Explorer:** Shows the database structure for AdventureWorks2019, including the BookstoreDB database.
- Query Editor:** Contains the same T-SQL code as the top window, followed by a new section:

```
580  
581  
582  
583     SELECT TOP 10 *  
584     FROM Reporting.ExecutionLog  
585     ORDER BY LogID DESC;
```
- Results:** Displays the execution log results in a table format:

LogID	ProcedureName	ExecutedSQL	ExecutedDate	ErrorMessage	ExecutedBy	HostName	DurationMS	ParameterValues	
1	3	Reporting.DynamicSalesSummarySecure	NULL	2025-11-16 20:08:25.117	RejectedInput: Suspicious keywords detected.	ANGELALEECCAB\janjan	ANGELALEECCAB	0	Northwest; DROP TABLE Sales.SalesOrderHeader ---;
2	2	Reporting.DynamicSalesSummaryVulnerable	SELECT	sp.Busine...	2025-11-16 20:08:20.977	NULL	641	Territory= 'OR' OR l=1 --; SalesPerson=N	
3	1	Reporting.SafeUpdateProductCost	UPDATE Production.Product...	2025-11-10 15:53:30.330	Product not found	NULL	NULL	NULL	NULL
- Messages:** Displays the message:

```
Query executed successfully.
```
- Completion Time:** 2025-11-16T23:53:30.330Z

3. Performance Comparison

I compared the execution time of:

- The secure procedure

- The vulnerable procedure

My results showed that the secure version performs better because parameterized SQL can reuse execution plans.

The vulnerable version usually takes longer because string-based SQL must be compiled again each time.

```

-- Audit Summary View
SELECT *
FROM Reporting.DSLAuditSummary;
GO

--test
EXEC Reporting.DynamicSalesSummarySecure
@Territory = 'Northwest',
@Category = 'Bikes';

EXEC Reporting.DynamicSalesSummaryVulnerable
@Territory = 'Northwest',
@Category = 'Bikes';

SELECT TOP 10 *
FROM Reporting.ExecutionLog
ORDER BY LogID DESC;

```

LogID	ProcedureName	ExecuteSQL	ExecutionTime	ErrorMessage	ProcOwner	DurationMS	ParameterValues
1	Reporting.DynamicSalesSummaryVulnerable	SELECT sp.Business...	2025-11-16 20:30:12.8287	NULL	ANGELALEECCAB\Junjun	125	Territory=Northwest; SalesPerson=NU
2	Reporting.DynamicSalesSummarySecure	SELECT sp.Business...	2025-11-16 20:30:12.8790	NULL	ANGELALEECCAB\Junjun	187	Territory=Northwest; SalesPerson=NU
3	Reporting.DynamicSalesSummarySecure	NULL	2025-11-16 20:08:25.117	RejectedInput: Suspicious keywords detected.	ANGELALEECCAB\Junjun	0	
4	Reporting.DynamicSalesSummaryVulnerable	SELECT sp.Business...	2025-11-16 20:02:50.9777	NULL	ANGELALEECCAB\Junjun	641	Territory= OR 1=>; SalesPerson=N
5	Reporting.SafeUpdateProductCost	UPDATE Production.Product...	2025-11-16 15:43:20.330	Product not found.	NULL	NULL	NULL

4. Audit Summary

I created a summary view ([Reporting.DSLAuditSummary](#)) to show:

- How many times each procedure was executed
- Average runtime
- Success and failure counts
- Number of rejected inputs

This makes it easier to analyze overall system behavior.

The screenshot shows the SSMS interface with the following details:

- Object Explorer:** Shows the connection to (localdb)\MSSQLLocalDB (SQL Server 15.0.4382 - ANGELA) and the AdventureWorks2019 database.
- SQL Query1.sq...Junjun (60)*:** The current query window contains the following T-SQL code:


```

SELECT
    ProcedureName,
    COUNT(*) AS ExecutionCount,
    AVG(DurationMS) AS AvgDurationMS,
    SUM(CASE WHEN ErrorMessage IS NULL
            THEN 1 ELSE 0 END) AS SuccessCount,
    SUM(CASE WHEN ErrorMessage IS NOT NULL
            THEN 1 ELSE 0 END) AS FailureCount,
    SUM(CASE WHEN ErrorMessage LIKE '%RejectedInput%'
            THEN 1 ELSE 0 END) AS RejectedInputCount
FROM Reporting.ExecutionLog
GROUP BY ProcedureName;
GO

--Audit Summary View
SELECT *
FROM Reporting.DSLAuditSummary;
      
```
- Results Grid:** Displays the output of the second part of the query, showing three rows of audit summary data:

ProcedureName	ExecutionCount	AvgDurationMS	SuccessCount	FailureCount	RejectedInputCount
Reporting.DynamicSalesSummarySecure	1	0	0	1	1
Reporting.DynamicSalesSummaryVulnerable	1	641	1	0	0
Reporting.SafeUpdateProductCost	1	NULL	0	1	0
- Status Bar:** Shows "Query executed successfully." and the session details: (localdb)\MSSQLLocalDB (15... ANGELA\EECCAB)Junjun (60) AdventureWorks2019 00:00:00 3 rows.

5. Findings and Recommendations

Findings

- The vulnerable procedure is easy to exploit.
- The secure procedure blocks injection successfully.
- Input validation adds an extra level of protection.
- Logging provides traceability and supports auditing.
- Parameterized SQL gives better performance.

Recommendations

1. Use `sp_executesql` for all dynamic SQL.
2. Add consistent input validation rules.
3. Log all DSL executions.
4. Restrict access to reporting procedures.
5. Review logs regularly for unusual activity.

6. Conclusion

Through this simulation, I learned the practical differences between secure and insecure Dynamic SQL.

By building both versions, I was able to observe:

- How SQL injection works
- How parameterization prevents attacks
- Why logging and validation are important
- Why secure SQL performs better

Overall, this project helped me understand how to build secure and reliable dynamic reporting procedures in SQL Server.