



CISC/CMPE 327 Software Quality Assurance

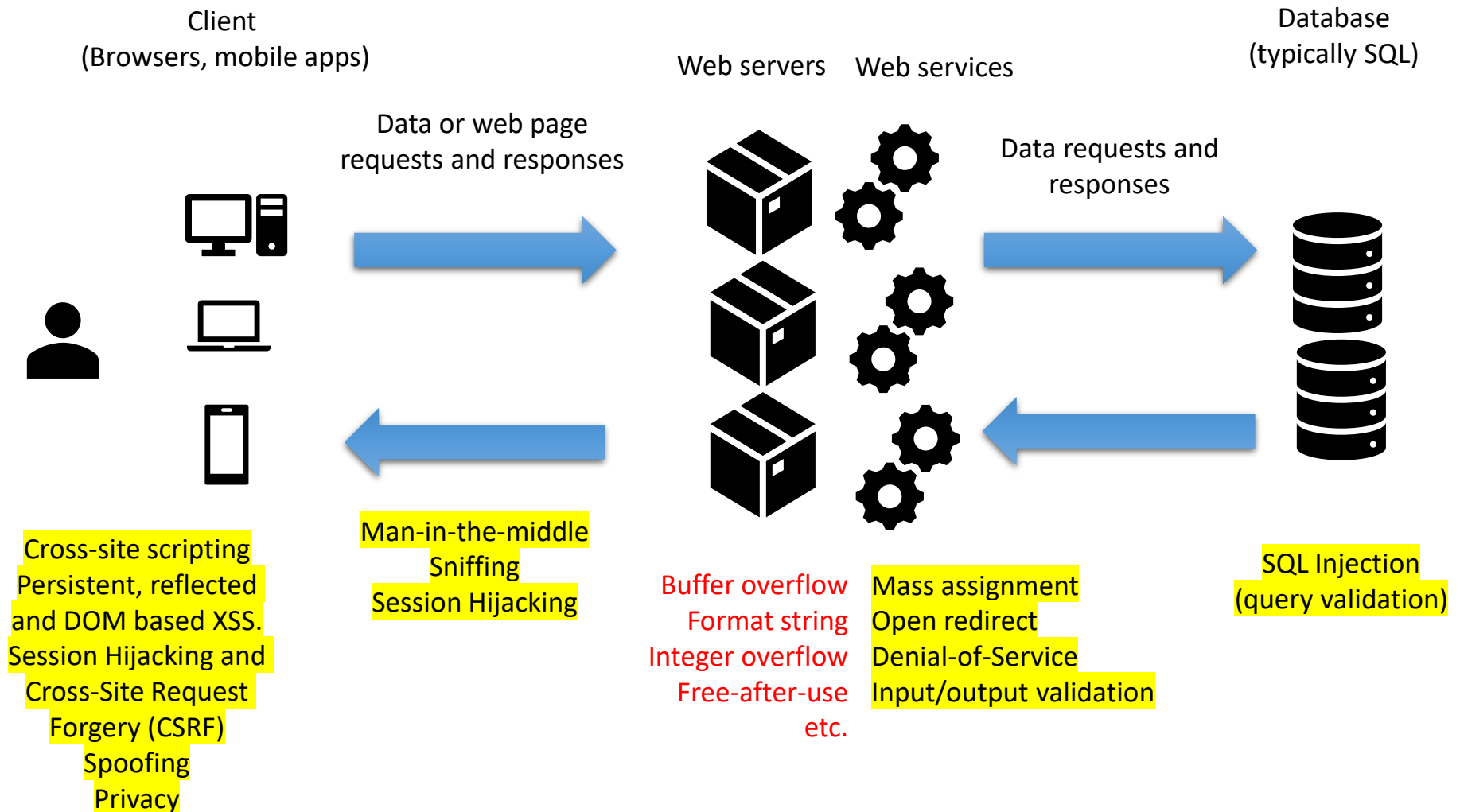
Queen's University, 2019–fall

Lecture #22 Web Application Security – SQL Injection

YESTERDAY

- CVE CVSS CWE
- How stack and stack frame work
- How/why does buffer overflow happen?
- Prevention
 - DEP
 - ASLR
 - Canary

Web application - vulnerabilities



OWASP Top 10 Application Security Risks—2017

A1:2017- Injection

A2:2017- Broken Authentication

A3:2017- Sensitive Data Exposure

A4:2017- XML External Entities (XXE)

A5:2017- Broken Access Control

A6:2017- Security Misconfiguration

A7:2017- Cross-Site Scripting (XSS)

A8:2017- Insecure Deserialization

A9:2017- Using Components with Known Vulnerabilities

A10:2017- Insufficient Logging & Monitoring

The Open Web Application Security Project

What is SQL?

- SQL (structured query language)
 - Standardized query language for requesting information from database. (ISO/IEC 9075:2016)
 - Supported by many database engines
 - MySQL, SQL Server, IBM DB2, Oracle, PostgreSQL, MS Access
 - Also known as relational databases.

SQL Data Model

- SQL models data as a relational table:

User ID	First_Name	Last_Name	Email	SIN
1	John	Taylor	JS@a.com	123-456-789
2	Adam	Taylor	AT@a.com	987-654-321

- Table contains a list of records. Each record corresponds to an entity.
- All records have the same set of attributes.
- Logically just like an excel table!

SQL - Select

- Retrieve specific attributes of one or more records:

User ID	First_Name	Last_Name	Email	SIN
1	John	Taylor	JS@a.com	123-456-789
2	Adam	Taylor	AT@a.com	987-654-321

SELECT SIN FROM users WHERE User_ID = '1'

123-456-789

SELECT SIN FROM users WHERE Last_Name = 'Talyor'

What is the output?

Database Popularity

Rank			DBMS	Database Model
Sep 2018	Aug 2018	Sep 2017		
1.	1.	1.	Oracle +	Relational DBMS
2.	2.	2.	MySQL +	Relational DBMS
3.	3.	3.	Microsoft SQL Server +	Relational DBMS
4.	4.	4.	PostgreSQL +	Relational DBMS
5.	5.	5.	MongoDB +	Document store
6.	6.	6.	DB2 +	Relational DBMS
7.	↑ 8.	↑ 10.	Elasticsearch +	Search engine
8.	↓ 7.	↑ 9.	Redis +	Key-value store
9.	9.	↓ 7.	Microsoft Access	Relational DBMS
10.	10.	↓ 8.	Cassandra +	Wide column store
11.	11.	11.	SQLite +	Relational DBMS
12.	12.	12.	Teradata +	Relational DBMS
13.	13.	↑ 16.	Splunk	Search engine
14.	14.	↑ 18.	MariaDB +	Relational DBMS
15.	15.	↓ 13.	Solr	Search engine

SQL

- Data Manipulation Language (DML)
 - Retrieve data from a database
 - Insert new data records
 - Delete exiting records
 - Update records
 - Support complex queries:
 - Averaged age of the students coming from Canada and the major is computer security
- Data Definition Language (DDL)
 - Create table
 - Delete table
 - Change table

SQL Injection

- The ability to **inject SQL commands** into **the database engine** through **an existing application**
(by The Open Web Application Security Project)
- SQL injection is a type of security exploit.
 - Attackers **manipulate** the **user input** forms.
 - Try to inject crafted SQL statements in to the inputs.
 - The **database engine didn't verify the input.**
 - Execute the attacker's injected statements.
 - Attacker gain resources or make change to data.

Application

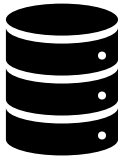
Web interface

Search Email Address

User ID



User input: 1



User ID	First_Name	Last_Name	Email	SIN
1	John	Smith	JS@a.com	123-456-789
2	Adam	Taylor	AT@a.com	987-654-321

SELECT Email FROM users WHERE User_ID = '1'

Email address is JS@a.com

Web interface



Attack

Web interface

Search Email Address

User ID



*User input: 1' UNION SELECT * FROM users --*



User ID	First_Name	Last_Name	Email	SIN
1	John	Smith	JS@a.com	123-456-789
2	Adam	Taylor	AT@a.com	987-654-321

*SELECT Email FROM users WHERE User_ID = '1' UNION SELECT * FROM users --'*

Web interface



Email address is JS@a.com *and Everything you have in the database.*

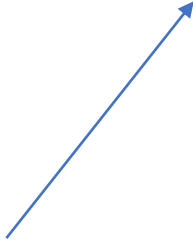
Attack Purposes

- Extract Data


SELECT Email FROM users WHERE User_ID = '1'

*SELECT Email FROM users
WHERE User_ID = '1' UNION SELECT * FROM users --'*

Added a ` to ends the string
delimiter



Use -- to comment out the `
added by the application

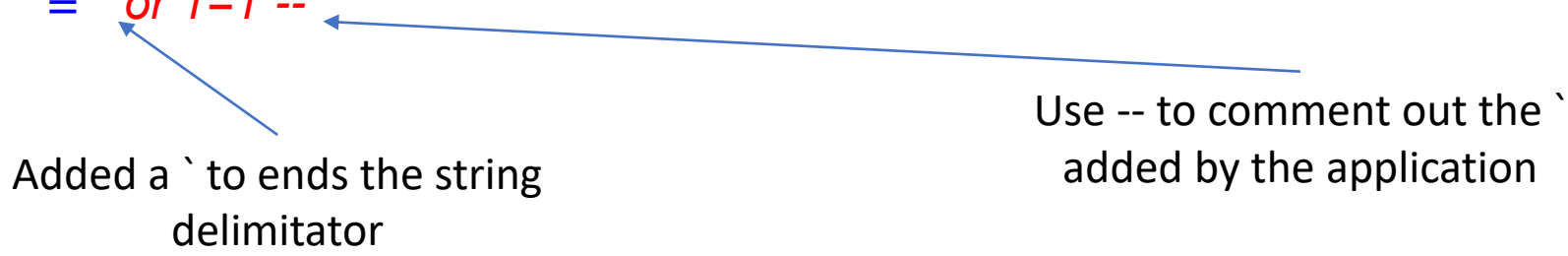


Attack Purposes

- Bypass Filters

```
SELECT *  
FROM users WHERE User_Name = 'alex' and Password  
= '01fac028dfb73'
```

```
SELECT *  
FROM users WHERE User_Name = 'alex' and Password  
= 'or 1=1 --'
```



Added a ` to ends the string
delimiter

Use -- to comment out the `
added by the application

Attack Purposes


- Modify Data

SELECT Email FROM users WHERE User_ID = '1'

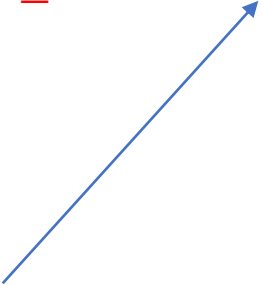
*SELECT Email FROM users
WHERE*

User_ID = "UPDATE Account Set Balance = 100000 WHERE User_ID = `1` --"

Added a ` to ends the string
delimiter



Use -- to comment out the `
added by the application



Attack Purposes

- Denial of Service
 - Drop table
 - Drop database
 - Delete backups
 - Lockup database
 - ' shutdown—
- Execute System Commands
 - Extended stored procedures in SQL Server enable one to execute shell command
 - *exec xp_regdeletekey*
 - Delete register keys
- Escalating the attack
 - Usually the database also contains the data for other applications

Attack Procedure

- Find injection bugs
 - Which user input field in the web application can we exploit?
- Fingerprinting the database
 - We need to know the available databases to be manipulated.
 - Try DDM and DDL queries.
- Fingerprinting the data tables
 - What tables are available and the attributes to exploit.
 - Try DDM and DDL queries.
 - `SELECT * FROM INFORMATION_SCHEMA.TABLES`

Finding injection bugs

- General Idea: try different SQL characters included as input to see if there is any different or error regarding the returned results.

Characters	Meaning in a SQL query
'	String delimitator
-- or #	Single line comment
/* ... */	Multi-line comment
+	String concatenation or addition
	String concatenation

Finding other tables

It is difficult if the user form does not display the result.
For example, a user login form.

Please enter your name and password

name:

password:

Submit Query

You must log in to proceed

Finding other tables

We can guess and verify if the table name exist by checking each characters:

```
' OR (SELECT COUNT(*) FROM  
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA  
LIKE '%a%')>1 --
```

Finding other tables

If the error message is hidden. Use timing attack.

```
' UNION select if(  
(SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES  
WHERE TABLE_SCHEMA LIKE '%a%')>1,  
sleep(5000), 'false')
```

Fingerprinting database

Checking the type of the database server.

The following SQL queries will only run without error in its corresponding type of server.

- Oracle:
 - `BITAND(1,1)-BITAND(1,1)`
- MS-SQL:
 - `@ @PACK_RECEIVED-@ @PACK_RECEIVED`
- MySQL:
 - `CONNECTION_ID()-CONNECTION_ID()`

Types of Attack

- 1st order injection:
 - Inject SQL statement into user input
 - Interpreted by the backend database engine
 - Executed immediately
 - Return the results.
- 2nd order injection:
 - Inject SQL statement into user input
 - (such as user name)
 - Stored it into the database by the application.
 - Will be interpreted and evaluated later.

Tools for Attacker

- SQLmap:
 - Automatic SQL injection and database takeover tool.
 - Automate the process of detecting and exploiting SQL injection flows.
 - Database/Table fingerprinting.
 - Wide range of supported database system.
- SQLsus
 - Another MySQL injection and takeover tool
 - Focus on speed and efficiency.
- SQLdict:
 - A dictionary attack tool for SQL Server
 - Heuristic brute force password guessing

SQL Injection (prevention)

Keyword blacklist filtering?

- Remove or disallow SQL keywords in user input.
 - If user name contains SELECT, tell the user that the name is not valid
 - Using a list of SQL keywords and characters.
- Does it work?

Evasion by String Manipulation

- Cases, space, and encoding
 - SeLeCT
 - %00SELECT
 - %53%45%4c%45%43%54
- Oracle database
 - `serv`||`ices` (|| denotes string concatenation)
- SQL Server
 - `serv`+`ices` (+ denotes string concatenation)
- MySQL
 - `ser` `vices`
- Using exec/execute in SQL
 - `exec(`SEL` + `ECT * F` + `ROM U` + `ers`)`
- Using variable in SQL
 - `set @x=0x12345677234234234; exec(@x)`

Evasion by Comments

- Comments are skipped:
 - `UNIO/**/N`
 - `SEL/*foo*\ECT`
 - `SEL/*foo*/ECT username, password FR/*foo*/OM users`

Evasion by Char()

- char(..) a function to decode numbers into its corresponding ASCII character

```
select ename, sal from emp where ename='marcus'
```

```
SELECT ename, sal FROM emp where ename=CHR(109)||CHR(97)||  
CHR(114)||CHR(99)||CHR(117)||CHR(115)
```

```
SELECT ename, sal FROM emp WHERE ename=CHAR(109)+CHAR(97)  
+CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)
```

Not even a single quote ` in the query!

Prevention

- Input validation and escaping
- White-list legitimate input, rather than black-list.
- IDS and IPS cannot guarantee a defense
- Least Privilege

Escaping

- Problem: simple string concatenation to construct the SQL query in the program:

```
Connection conn = DriverManager.getConnection(myUrl, "root", "");  
// our SQL SELECT query.
```

```
// create the query:  
String query = "SELECT * FROM users WHERE user_id = `" + user_id + "`";
```

```
// create the java statement  
Statement st = conn.createStatement();
```

```
// execute the query, and get a java resultset  
ResultSet rs = st.executeQuery(query);
```

Need to tell the database that user_id is a string, regardless what the user_id is there.

Escaping

- Don't trust any input.
- Escape special characters, so the database knows that it is user input instead of a query keyword.

*' UNION SELECT * FROM users --*

SELECT Email FROM users
*WHERE User_ID = '1' UNION SELECT * FROM users --'*

SELECT Email FROM users
*WHERE User_ID = '1\' UNION SELECT * FROM users \-\'*



Interpreted as a user input string, rather than SQL keywords

Escaping – Don't do it yourself!

- Escaping is a complex task:
 - Escaped apostrophe ?
 - But the attacker can use %27 (php)
- OWASP ESAPI
 - The OWASP Enterprise Security API
 - A free, open source, web application security control library that makes it easier for programmers to write lower-risk applications.
 - `ESAPI.encoder().encodeForSQL(..)`
- Most database engine API comes with:
 - Stock SQL escaping API.
 - Parameterized query, a way to fill in parameters in SQL query.
 - Separate the data definition and the SQL syntax.

Escaping – Don't do it yourself!

- Parameterized query, a way to fill in parameters in SQL query.
 - Separate the data definition and the SQL syntax.

```
Connection conn = setupTheDatabaseConnectionSomehow();
```

```
PreparedStatement stmt = conn.prepareStatement(  
    "SELECT * FROM user where user_id = {}"  
);
```

```
stmt.setString(1, name_id);
```

```
stmt.execute();
```

Escaping – Don't do it yourself!

- Parameterized query, a way to fill in parameters in SQL query.
 - Should be used for every database query.
 - All user input fields need to be parameterized.
 - Metadata such as database name and table name should not be parametrized.
 - SQL keywords should not be parameterized.
 - Keep your database API library up-to-date.

Escaping – Don't do it yourself!

- Escaping is a complex task:
 - Escaped apostrophe ?
 - But the attacker can use %27 (php)
- Most database engine API comes with:
 - Stock SQL escaping API.
 - Templating, a way to fill in parameters in SQL query.
 - Separate the data definition and the SQL syntax.
- OWASP ESAPI
 - The OWASP Enterprise Security API
 - A free, open source, web application security control library that makes it easier for programmers to write lower-risk applications.
 - `ESAPI.encoder().encodeForSQL(..)`

White-list legitimate input

- Define what is valid, rather than what is invalid.
- Limit length!
- Email
 - Define the required format of an email address.
- User name
 - Define what is allowed to be used as user name.
- Password
 - Allowed characters and required complexity.
- Number
 - Make sure they are actually only numbers.
 - Check range. Age = 65589 ? Suspicious!
 - Alternatively, avoid numbers.
- Verify on server, not just on client!

IDS and IPS

- Intrusion Detection System
- Intrusion Prevention System
- Used on system level to detect or prevent intrusion.
- Detection on SQL injection relies on signatures.
 - Signatures on the SQL query: `` or 1=1 -`
 - Can be evaded:
 - ``/**/o/**/r/**/ 1 /**/=/**/1/**/-`
 - `UNI*this is a comment*\ON`
- They should be never used alone to protect applications from SQL injection.
- Need to protect it at the application level.

Least Privilege

- Database engine should run using a dedicated user account on the server.
- Minimize the privileges for every database accounts.
 - Different account used for different application.
 - For example, an account used in web application shouldn't have the write access on the database account table.
 - Generally, except administrator account, no application account can create or modify the metadata (databases, tables..)

Least Privilege

- Most database comes with lots of default functionalities that are not used by your application.
- But they provide the attackers additional attack surface to be leveraged.
- Keep your database service updated.

Prevention

Fundamentally, it is the role of application to work against the SQL injection attacks.