

***Software Security***

**“SQL Injection”**

**Lecturer contact :**

**Dr. Drake Patrick Mirembe, Email:** [**drake.mirembe@mak.ac.ug**](mailto:drake.mirembe@mak.ac.ug) **Website:** [**www.drakemirembe.org**](http://www.drakemirembe.org/)

**Course resources: https://muele.mak.ac.ug**

What we are going to cover



* ***SQL injection***
  + What is SQL?
  + What is SQL injection?
  + Programming languages vulnerable to SQL injection
  + Examples of SQL injection
  + How to reduce risks of SQL injection
  + Summary

# What is SQL



* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

# SQL injection



* SQL injection is an attack that consists of insertion or “injection” of a SQL query via the input data from the client to the application.
* A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS).

# SQL injection



* SQL injection is a code injection technique that might destroy your database.
* SQL injection is one of the most common web hacking techniques.
* SQL injection is the placement of malicious code in SQL statements, via web page input.

# SQL injection



* *…continuation*
* SQL injection usually occurs when you ask a user for input, like their username/user\_id, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

**Programming languages vulnerable to SQL injection**



* Any programming language used to interface with a database can be affected.

## Higher-level programming languages affected by SQL injection



* Perl
* Python
* Ruby
* Java
* ASP
* ASP.NET
* JSP
* PHP
* C#
* VB.NET

**Lower-level programming languages affected by SQL injection**



* C
* C++

**Reference**



* Michael Howard, 2010, 24 DEADLY SINS OF SOFTWARE SECURITY – Page 6

**Examples of SQL injection**



* SQL Injection Based on 1=1 is Always True
* SQL Injection Based on ""="" is Always True
* SQL Injection Based on Batched SQL Statements
* SQL Injection Based on Batched SQL comment block

## SQL Injection Based on 1=1 is Always True



SELECT \* FROM Users WHERE UserId = 200 OR 1=1;

## SQL Injection Based on 1=1 is Always True



SELECT UserId, Name,

Password FROM Users WHERE UserId = 105 or 1=1;

## SQL Injection Based on ""="" is Always True



$ uName = $\_GET("username");

$uPass = $\_GET("userpassword");

sql = 'SELECT \* FROM Users WHERE Name ="' + $ uName + '" AND Pass ="' + $uPass + '"'

## SQL Injection Based on Batched SQL Statements



SELECT \* FROM Users; DROP TABLE Suppliers

## SQL Injection Based on Batched SQL comment block



SELECT \* FROM Users; DROP TABLE Suppliers

## SQL Injection Based on Batched SQL comment block



SELECT \* FROM Users; DROP TABLE Suppliers

' OR '1'=‘1’ --

' OR '1'='1’ {

' OR '1'='1' /\*

**Mitigation/Protection against SQL Injection**



* Use of SQL parameters
* Detection of malicious characters
* Escaping
* Pattern check
* Database permissions

## Use of SQL parameters (Prepared Statements)



* A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

## Use of SQL parameters (Prepared Statements)



1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

## Use of SQL parameters (Prepared Statements)



$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");

$stmt->bind\_param("sss", $firstname,

$lastname, $email);

$stmt->execute( );

## Detection of malicious characters



* SQL injection filtering works in similar way to emails spam filters.
* Database firewalls detect SQL injections based on the number of invalid queries from host, the presence of OR and UNION blocks inside of the request, or other characteristics.

## Escaping



* A popular, though error-prone and ultimately doomed way to prevent injections is to attempt to escape all characters that have a special meaning in SQL.
* The manual for an SQL DBMS explains which characters have a special meaning, which allows creating a comprehensive blacklist of characters that need translation.

## Escaping



* For instance, every occurrence of a single quote (') in a parameter must be replaced by two single quotes ('') to form a valid SQL string literal.
* For example, in PHP it is usual to escape parameters using the

function mysqli\_real\_escape\_string(); before sending the SQL query:

## Pattern check



* Integer, float or boolean, string parameters can be checked if their value is valid representation for the given type.
* Strings that must follow some strict pattern (date, UUID, alphanumeric only, etc.) can be checked if they match this pattern.

## Database permissions



* Limiting the permissions on the database login used by the web application to only what is needed may help reduce the effectiveness of any SQL injection attacks that exploit any bugs in the web application.

## Database permissions



* deny select on sys.sysobjects to webdatabaselogon; deny select on sys.objects to webdatabaselogon;
* deny select on sys.tables to webdatabaselogon;
* deny select on sys.views to webdatabaselogon;
* deny select on sys.packages to webdatabaselogon;

Summary



* Do understand the database you use. Does it support stored procedures? What is the comment operator? Does it allow the attacker to call extended functionality?
* Do understand common SQL injection attack methods against the database you use.
* Do check the input for validity and trustworthiness.
* Do check for input validity at the server.
* Do use parameterized queries, also known as prepared statements, placeholders, or parameter binding, to build SQL statements.

Summary



* Do store the database connection information in a location outside of the application, such as an appropriately protected configuration file or the Windows registry.
* Do encrypt sensitive database data.
* Do not trust input used to build SQL statements.
* Do not use string concatenation to build SQL statements unless there is absolutely no other way to build a SQL statement a safely.
* Do not execute untrusted parameters within stored procedures.
* Do not check for input validity only at the client.

THANK YOU!



