SQL Injection

SQL Injections

SQL injection is a particularly widespread and dangerous form of injection attack that consists of **injecting SQL commands into the database engine** through an existing application

Relational Databases

- A relational database contains one or more relations (i.e., tables)
  - Each table is identified by a name
  - Each table has a fixed number of named and *typed* columns
- Tables contain records (rows) with data

userID	Name	LastName	Login	Password
1	John	Smith	jsmith	hello
2	Adam	Taylor	adamt	qwerty
3	Daniel	Thompson	dthompson	dthompson

Structured Query Language (SQL)

- SQL is a data manipulation language (DML) to access databases and can
  - Query the content of a database (`SELECT`)
  - Modify data in a database:
    - `Insert` add rows
    - `Update` modify rows
    - `Delete` remove rows
- SQL is standard (ANSI and ISO) but most DBMS implement language extensions in addition to the standard

SQL Data Definition Language (DDL)

- SQL DML operates ***on data*** in relations
- DDL defines and modifies the ***structure of relations*** in the database
  - {CREATE,ALTER,DROP} TABLE
  - Assign types to columns
    e.g., `INT`, `CHAR`, `geography` (ellipsoidal spatial)
  - Default values
  - Referential integrity
  - Constraints (`NOT NULL`, `UNIQUE`, etc.)
- DDL and DML parsed by the same SQL engine

SQL – SELECT Definition

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [,
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]

where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | colu
    function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
```

SQL Example

To extract the last name of a user from the previous table

```
mysql> SELECT LastName FROM users WHERE UserID = 1;
+----------------+
| LastName       |
+----------------+
| Smith          |
+----------------+
1 row in set (0.00 sec)
```

SQL Example

Extract information on user based on username + password (e.g., to perform authentication during login)

```
mysql> SELECT * FROM users WHERE login =‘jsmith‘ AND
password = ‘hello‘;
+--------+------+----------+--------+----------+
| UserID | Name | LastName | Login  | Password |
+--------+------+----------+--------+----------+
| 1      | John | Smith    | jsmith | hello    |
+--------+------+----------+------- +----------+
1 row in set (0.00 sec)
```

SQL Injections

- To exploit a SQL injection flaw, the attacker must find a parameter that the web application uses to construct a database query
- By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database
- The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents

SQL Injections

- Not a DB or web server problem
  It is a flaw in the web application!
  - Many programmers are still not aware of this problem
  - Many of the tutorials and demo “templates” are vulnerable
  - Even worse, many of solutions posted on the Internet are not good enough

Simple SQL Injection Example

Perl script looks up *username* and *password*

```
…
$query = new CGI;
$username = $query->param(“username”);
$password = $query->param(“password”);
…
$sql_command = “select * from users where
username=‘$username’ and password=‘$password’”;
$sth = $dbh->prepare($sql_command)
…
```

No Validation!

Simple SQL Injection Example

- If the user enters a ‘ (single quote) as the password, the SQL statement in the script would become:
  - `select * from users where login =‘ ‘ and password = ‘‘‘`
  - An SQL error message would be generated

- If the user enters (injects): ‘ or login =‘jsmith as the password, the SQL statement in the script would become:
  - `select * from users where login =‘ ‘ and password = ‘‘ or login = ‘jsmith‘`
  - Hence, a *different* SQL statement has been injected than what was originally intended by the programmer!

Obtaining Information using Errors

- Errors returned from the application might help the attacker (e.g., ASP – default behavior)
  - Username: ' union select sum(id) from users

    Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

    /process_login.asp, line 35
- Make sure that you do not display unnecessary debugging and error messages to users.
  - For debugging, it is always better to use log files (e.g., error log).

Some SQL Attack Examples

- `select * …;`
  `insert into user values(“user”,”h4x0r”);`
  - Attacker inserts a new user into the database
- Call “stored procedures” (e.g., in SQL Server)
  - `xp_cmdshell()` → arbitrary command execution
  - “bulk insert” statement to read any file on the server
  - e-mail data to the attacker’s mail account
  - Play around with the registry settings
- `select *… ; drop table SensitiveData;`
- Appending “`;`” character does not work for all databases. Might depend on the driver (e.g., MySQL)

DEMO
Simple SQL Injection

Advanced SQL Injection

- Web apps often escape the ‘ and “ (e.g., in PHP)
  - Will prevent most SQL injection attacks… but there might still be vulnerabilities
- Database columns have ***types***
  - ‘ or “ characters not necessary (e.g., … where id=1)
- Attacker might still inject strings into a database by using the `char` function (e.g., SQL Server):
  - insert into users values(666,char(0x63)+char(0x65)…)

Blind SQL Injection

- Typical countermeasure: Don't display error messages. But, is this enough?
  - No, your application may still be vulnerable to *blind SQL injection*
- Example: Suppose there is a news site
  - Press releases are accessed with pressRelease.jsp?id=5
  - An SQL query is created and sent to the database:

    select title, description FROM pressReleases where id=5;
  - Any error messages are smartly filtered by the application

Blind SQL Injection

- How can we inject statements into the application and exploit it?
  - We do not receive feedback from the application so we can use a trial-and-error approach
  - First, we try to inject pressRelease.jsp?id=5 AND 1=1
  - The SQL query is created and sent to the database:

  select title, description FROM pressReleases where id=5 AND 1=1
  - If there is an SQL injection vulnerability, the *same* press release should be returned
  - If input is validated, id=5 AND 1=1 should be treated as value

Blind SQL Injection

- When testing for vulnerability, we know 1=1 is always true
  - However, when we inject other statements, we do not have any information
  - What we know: If the same record is returned, the statement must have been true
  - For example, we can ask server if the current user is “h4x0r”:

pressRelease.jsp?id=5 AND user_name()=‘h4x0r’

  - By combining subqueries and functions, we can ask more complex questions (e.g., extract the name of a database character by character)

SQL Injection Solution

- Instead of string-building SQL, call stored procedure (e.g., in Java):

```
CallableStatements cs =
    dbConnection.prepareCall("{call
    getPressRelease(?)}");
cs.setInt(1,Integer.parseInt(request.getParameter(
    "id")));
ResultSet rs = cs.executeQuery();
```

- In ASP.NET, there is a similar mechanism

Exploits Of A Mom