SQL Injection

The ability to inject SQL commands into the database engine through an existing application

What is SQL?

- SQL stands for **Structured Query Language**
- Allows us to access a database
- ANSI and ISO standard computer language
  - The most current standard is SQL99
- SQL can:
  - execute queries against a database
  - retrieve data from a database
  - insert new records in a database
  - delete records from a database
  - update records in a database

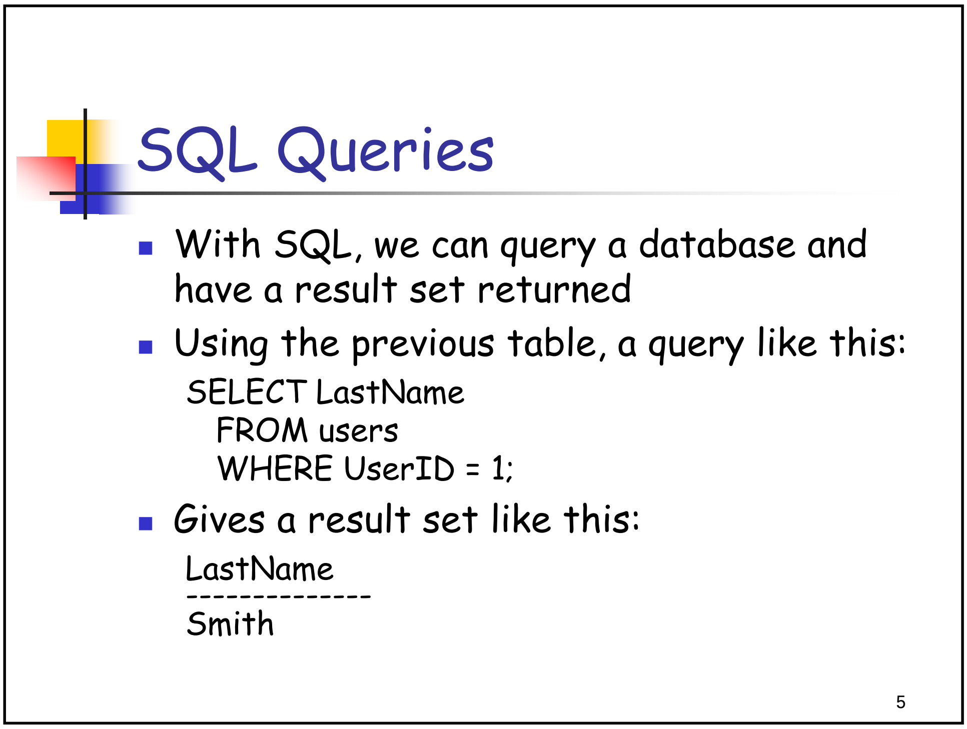
SQL is a Standard - but...

- There are many **different versions** of the SQL language
- They support the same major **keywords** in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).
- Most of the SQL database programs also have their own **proprietary extensions** in addition to the SQL standard!

SQL Database Tables

- A relational database contains one or more tables identified each by a name
- Tables contain records (rows) with data
- For example, the following table is called "users" and contains data distributed in rows and columns:

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



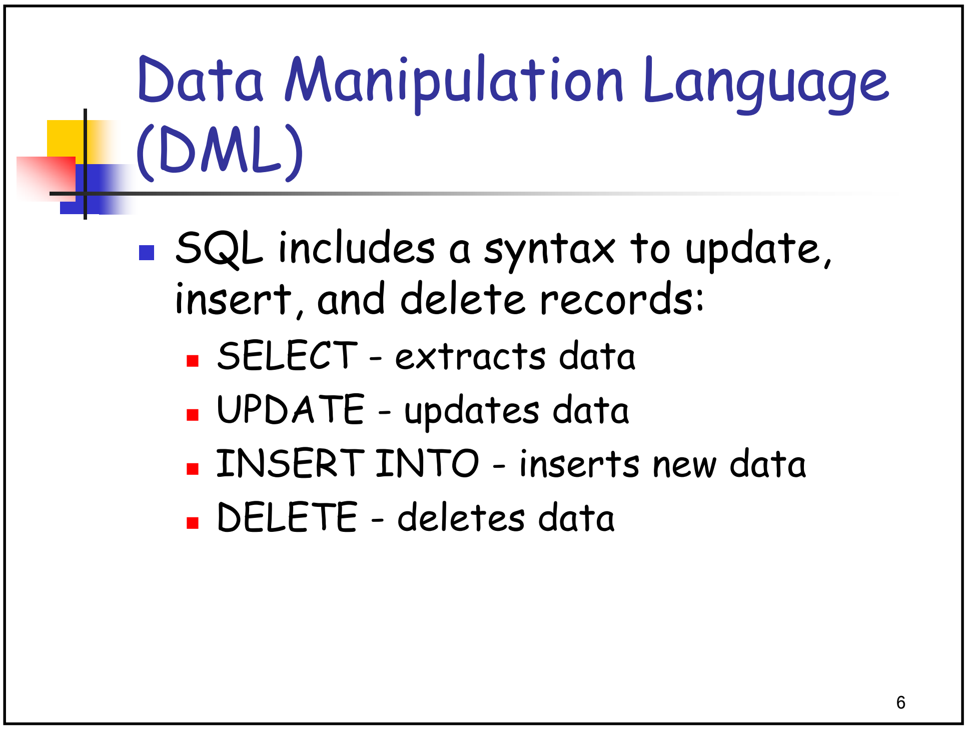
SQL Queries

- With SQL, we can query a database and have a result set returned
- Using the previous table, a query like this:

```
SELECT LastName
   FROM users
   WHERE UserID = 1;
```

- Gives a result set like this:

```
LastName
---------------
Smith
```



Data Manipulation Language (DML)

- SQL includes a syntax to update, insert, and delete records:
  - SELECT - extracts data
  - UPDATE - updates data
  - INSERT INTO - inserts new data
  - DELETE - deletes data

Data Definition Language (DDL)

- The Data Definition Language (DDL) part of SQL permits:
  - Database tables to be created or deleted
  - Define indexes (keys)
  - Specify links between tables
  - Impose constraints between database tables
- Some of the most commonly used DDL statements in SQL are:
  - CREATE TABLE - creates a new database table
  - ALTER TABLE - alters (changes) a database table
  - DROP TABLE - deletes a database table

How common is SQL injection?

- It is probably the most common Website vulnerability today
- It is a flaw in "web application" development, it is not a Database or web server problem
  - Most programmers are still not aware of this problem
  - Many tutorials and demo "templates" are vulnerable
  - Even worse, a lot of solutions posted on the Internet are not good enough

Vulnerable Applications

- Almost all SQL databases and programming languages are potentially vulnerable
  - MS SQL Server, Oracle, MySQL, Postgres, DB2, MS Access, Sybase, Informix, etc
- Accessed through applications developed using:
  - Perl and CGI scripts that access databases
  - ASP, JSP, PHP
  - XML, XSL and XSQL
  - Javascript
  - VB, MFC, and other ODBC-based tools and APIs
  - DB specific Web-based applications and API's
  - Reports and DB Applications
  - 3 and 4GL-based languages (C, OCI, Pro*C, and COBOL)
  - …

How does SQL Injection work?

Common vulnerable login query

```
SELECT * FROM users
WHERE login = 'victor'
AND password = '123'
```

(If it returns something then login!)

ASP/MS SQL Server login syntax

```
var sql = "SELECT * FROM users
WHERE login = '" + formusr + "'
AND password = '" + formpwd + "'";
```

Injecting through Strings

formusr = **' or 1=1 – –**
formpwd = anything

Final query would look like this:

SELECT * FROM users
WHERE username = ' ' **or 1=1**
– – AND password = 'anything'

The power of '

- It closes the string parameter
- Everything after is considered part of the SQL command
- Misleading Internet suggestions include:
  - Escape it : replace ' with ''
- String fields are very common but there are other types of fields:
  - Numeric
  - Dates

If it were numeric?

```
SELECT * FROM clients
WHERE account = 12345678
AND pin = 1111
```

PHP/MySQL login syntax

```
$sql = "SELECT * FROM clients WHERE " .
"account = $formacct  AND " .
"pin = $formpin";
```

Injecting Numeric Fields

$formacct = **1 or 1=1 #**

$formpin = 1111

Final query would look like this:

```
SELECT * FROM clients
WHERE account = 1 or 1=1
# AND pin = 1111
```

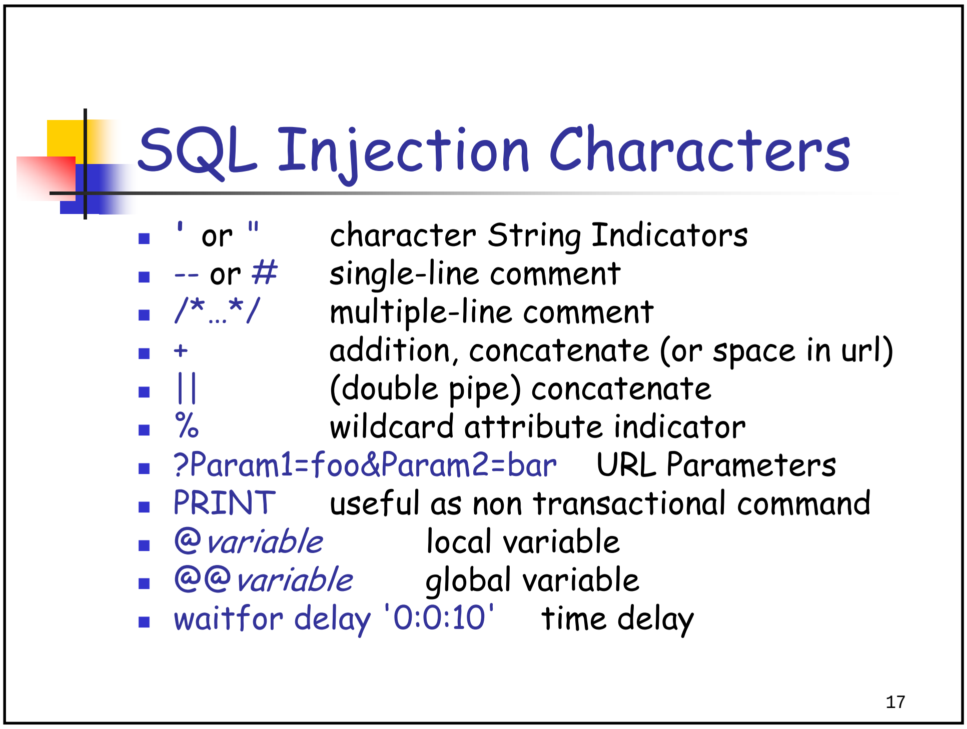
Evasion Techniques

- Input validation circumvention and IDS Evasion techniques are very similar and rely on "signatures"
- Signatures can be evaded easily
- Input validation, IDS detection AND strong database and OS hardening must be used together

IDS Signature Evasion

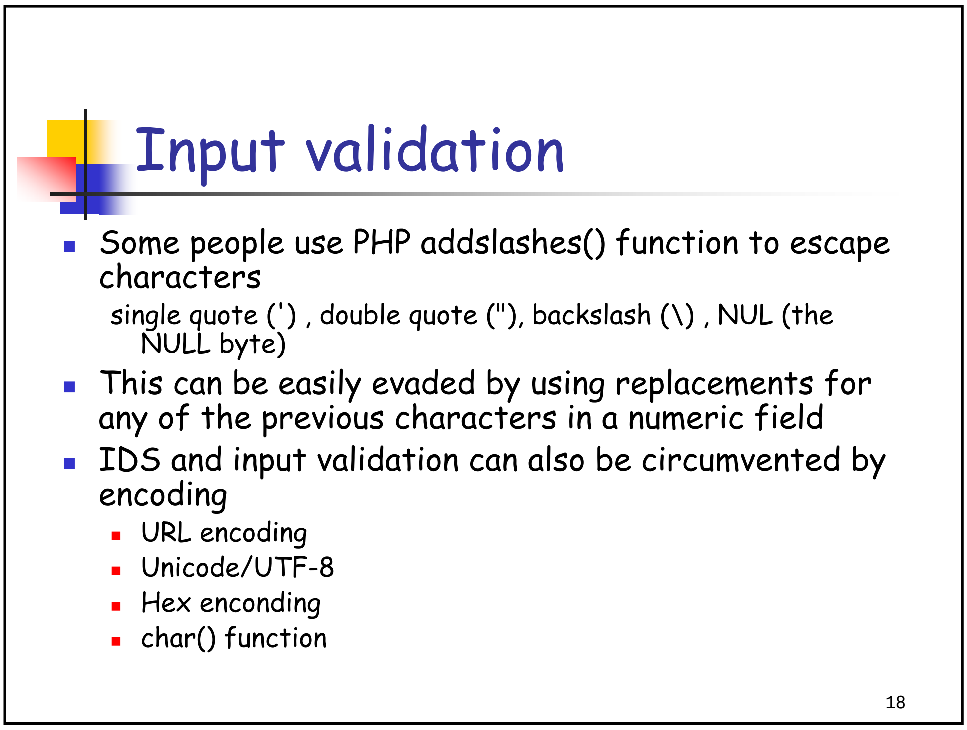
Evading ' OR 1=1 signature

- ' OR 'unusual' = 'unusual'
- ' OR 'something' = 'some'+'thing'
- ' OR 'text' = N'text'
- ' OR 'something' like 'some%'
- ' OR 2 > 1
- ' OR 'text' > 't'
- ' OR 'whatever' IN ('whatever')
- ' OR 2 BETWEEN 1 AND 3



SQL Injection Characters

' or "	character String Indicators
-- or #	single-line comment
/*…*/	multiple-line comment
+	addition, concatenate (or space in url)
||	(double pipe) concatenate
%	wildcard attribute indicator
?Param1=foo&Param2=bar	URL Parameters
PRINT	useful as non transactional command
@*variable*	local variable
@@*variable*	global variable
waitfor delay '0:0:10'	time delay



Input validation

- Some people use PHP addslashes() function to escape characters

  single quote (') , double quote ("), backslash (\) , NUL (the NULL byte)
- This can be easily evaded by using replacements for any of the previous characters in a numeric field
- IDS and input validation can also be circumvented by encoding
  - URL encoding
  - Unicode/UTF-8
  - Hex enconding
  - char() function

MySQL Input Validation Circumvention using Char()

- Inject without quotes (string = "%"):
  - ' or username like char(37);
- Inject without quotes (string = "root"):
  - ' union select * from users where login = char(114,111,111,116);

Defending against SQL injections

- Sanitize all input.
  - Assume all input is harmful.
  - Validate user input that contains dangerous keywords or SQL characters, such as “xp_cmdshell”, “- -”, and “;”.
  - Consider using regular expressions to remove unwanted characters. This approach is safer than writing your own search and replace routines.
- Run with least privilege.
  - Do not execute an SQL SELECT statement as “sa”. Create low-privilege accounts to access data.
  - Use SQL permissions to lock down databases, stored procedures, and tables.
  - Remove unused stored procedures.

Defending against SQL injections

- Do not allow clients to view ODBC/OLE DB error messages. Handle these errors with your own code. By default, ASP pages returns error messages to clients.
- Enable logging of all user access, and set alerts to log all failed attempts to access objects.
- Do not use string concatenations to build SQL queries. Instead, use parameterized queries or **parameterized stored procedures**, because they explicitly define input and output values and do not process multiple statements as a batch.

Back to a previous example

```
var sql = "SELECT * FROM users
 WHERE login = '" + formusr + "'
 AND password = '" + formpwd + "'";
```

is replaced by

```
SqlConnection objConnection=new SqlConnection(_ConnectionString);
objConnection.Open();
SqlCommand objCommand = new SqlCommand( "SELECT * FROM User WHERE
   login = @Name AND password = @Password", objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PasswordTextBox.Text);
SqlDataReader objReader = objCommand.ExecuteReader();
if (objReader.Read())
   { ...
```

Why is it safer? Because the SQL server knows that the value of the parameter is not actual code to execute, but data