Lecture 8 – Database Security

University of Illinois

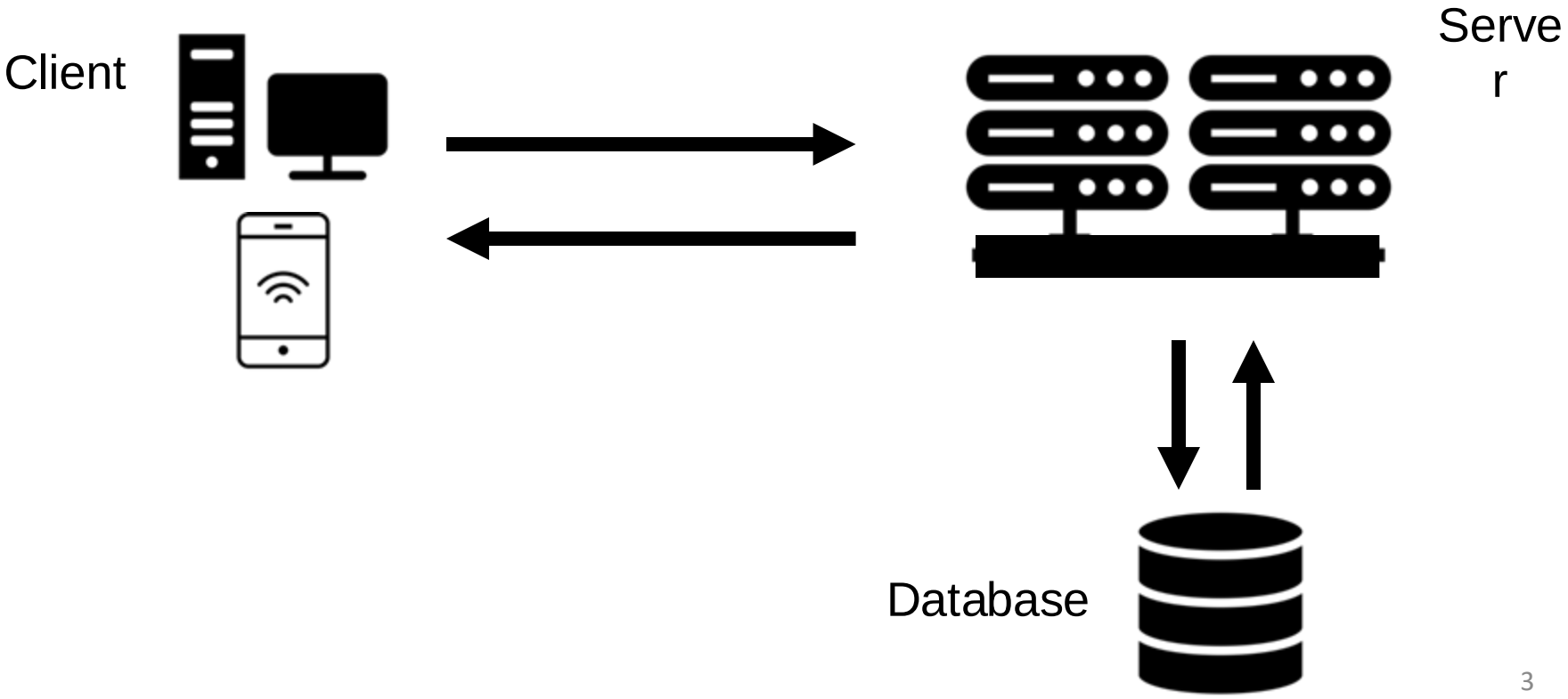
ECE 422/CS 461

Goals

- By the end of this lecture you should:
  - Be able to execute SQL injection attacks
  - Be able to explain the principles of (SQL) injection attacks and defenses
  - Know basic mechanisms in databases access control

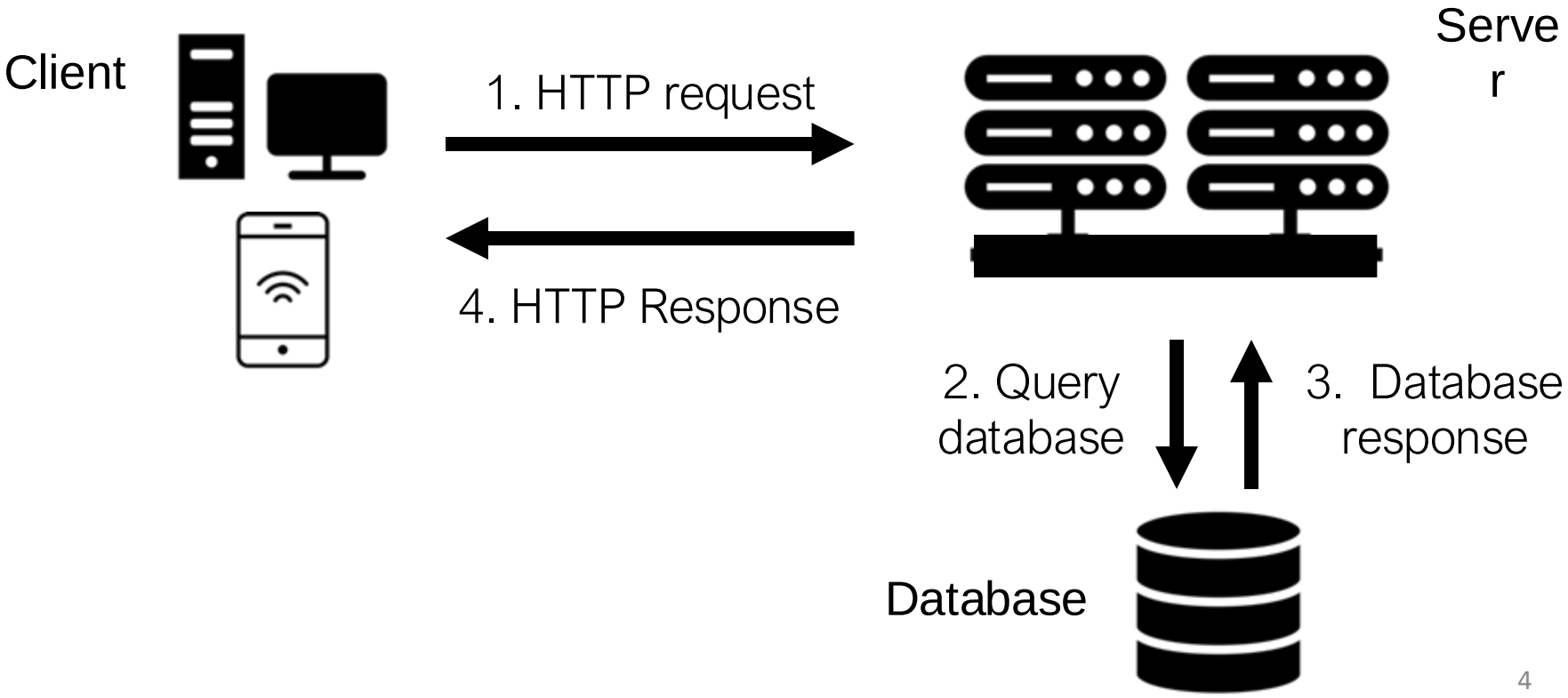
Three-Tiered Web Apps

- Most websites need to store data
  - e.g., account info, merchandise info



Typical Workflow

- Server retrieves data to answer HTTP queries



Relational database management systems (RDBMS)

- Fundamental technology for managing structured data
- Data organized as **tables**

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Structured Query Language (SQL)

- Language used to manage data in RDBMS
  - Most widely used database language
  - Common queries: SELECT, INSERT, DELETE, ...
- Standardized by ANSI in 1986 and ISO in 1987
  - However, syntax differ slightly across vendors …
  - We use MySQL in this class

Example SQL Query

SELECT * FROM Employees

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Example SQL Query

```
SELECT * FROM Employees
WHERE LastName = 'Skywalker'
```

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Example SQL Query

SELECT Phone, Email FROM Employees

WHERE LastName = ‘Skywalker’

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Example SQL Query

```
SELECT Phone, Email FROM Employees
WHERE LastName = 'Skywalker'
      OR Email = 'pi_delicious@gmail.com'
```

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Breaking it down

- SELECT – statement/command to select columns from a table
  - * means all columns
- FROM – identifies the table
- WHERE – identify rows to be selected
  - Support comparisons: =, <, <=, >, >=, <>
  - Support arithmetic ops: +, -, *, /, %
  - Support logical operators: AND, OR, NOT

Exercise: What is Selected?

SELECT Phone, Email FROM Employees

WHERE LastName = 'Skywalker' OR Email <> ''

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Exercise: What is Selected?

SELECT Phone, Email FROM Employees

WHERE LastName = 'Skywalker' OR Email <> ''

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Exercise: What is Selected?

SELECT Phone, Email FROM Employees

WHERE LastName = 'Skywalker' OR 2 <= 3

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

Exercise: What is Selected?

SELECT Phone, Email FROM Employees

WHERE LastName = 'Skywalker' OR 2 <= 3

FirstName	LastName	Phone	Email
Holden	Caufield	(217)-555-3251	nophoney@hotmail.com
Richard	Parker	(217)-555-1212	pi_delicious@gmail.com
Hucklberry	Finn	(217)-555-8519	raftboy@hotmail.com
Luke	Skywalker	(217)-555-2917	wompratbullseye@gmail.com
Bella	Swan	(217)-555-6666	edwardsgrrl04@aol.com
Marty	McFly	(217)-555-1987	delorian88@gmail.com

Employees

More SQL Queries

- Other common SQL keywords/commands https://www.w3schools.com/sql/

- INSERT INTO: add rows to a table
- UPDATE: modify existing rows in a table
  - SET identifies columns and WHERE identifies rows
- DELETE FROM: delete rows from a table
  - WHERE identifies rows

More SQL Queries

- INSERT INTO Employees VALUES (‘Erica’, ‘Logan’, ‘217-555-9145’, ‘erica@logan.net’)

- UPDATE Employees SET phone = ‘217-342-8631’ WHERE LastName = ‘Skywalker’

- DELETE FROM Employees
  WHERE LastName = ‘Skywalker’

More SQL Queries

- CREATE: create a new table
  - Specify a list of columns and their data types

- DROP: delete an entire table

```
DROP TABLE Employees
```

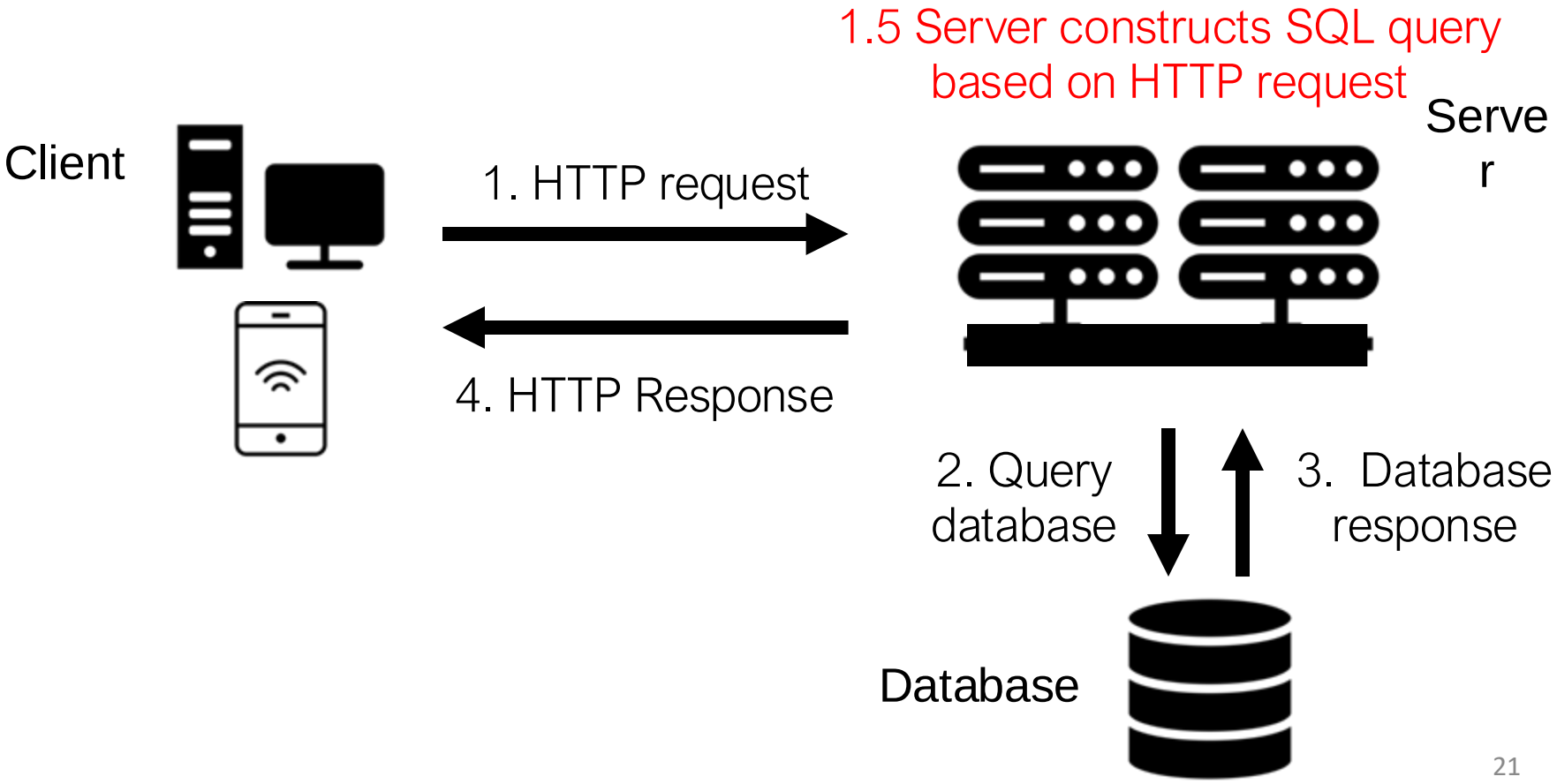
More SQL Syntax

- -- (two hashes): single-line comment
  - Similar to // in C or # in Python
- ; (semicolon): ends a query
  - Can be omitted for a single query
  - Must be added when having two or more queries
- ‘’ (single quotes) for string, but “” also work
- Unmatched () ‘’ “” usually give errors

SQL Injection

Typical Workflow

- Server retrieves data to answer HTTP queries



Web + SQL Example

- Suppose server takes HTTP requests and uses the following PHP code to construct SQL queries

```
$user = $_POST[‘User’];
$sql = "SELECT * FROM Orders WHERE Recipient =
   ‘$user’  ";
```

Web + SQL Example

$user = $_POST[‘User’];

$sql = "SELECT * FROM Orders WHERE Recipient = ‘**$user**’ ";

- User sends HTTP request: POST User=**Bob**
- Web server issues SQL query

SELECT * FROM Orders

WHERE Recipient = ‘**Bob**’

SQL Injection

- (Yet another) common and damaging attack
  - User inputs data in a web form
  - User input is not properly sanitized
  - User input is used as part of SQL query
  - Clever input can manipulate SQL query behavior

SQL Injection Example

$user = $_POST[‘User’];

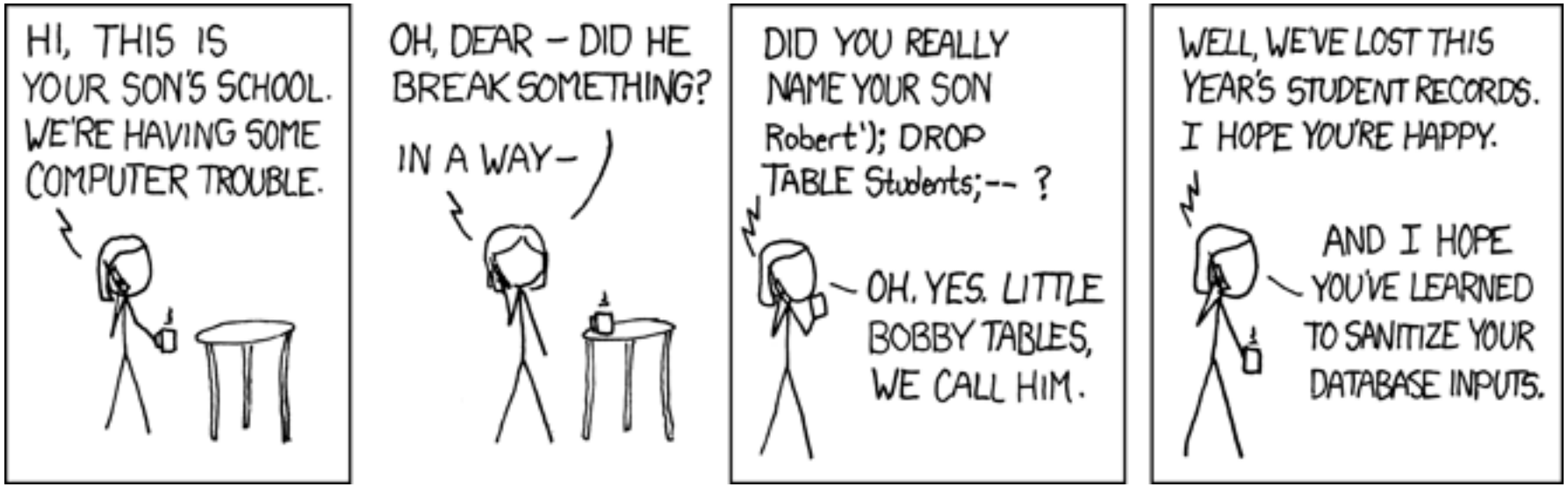
$sql = "SELECT * FROM Orders WHERE Recipient = ‘**$user**’ ";

- User POST User=**Bob’ OR 1=1; --**
- Web server issues SQL query

SELECT * FROM Orders

WHERE Recipient = ‘**Bob’ OR 1=1;** -- ’

Meet Bobby Tables



SQL Injection Example

$user = $_POST[‘User’];

$sql = "SELECT * FROM Orders WHERE Recipient = ‘**$user**’ ";

- User POST Recipient=**Bob’; DROP TABLE Orders; --**
- Web server issues SQL query

SELECT * FROM Orders WHERE Recipient = ‘**Bob’; DROP TABLE Orders; --** ’

SQL Injection Defense

- Make sure data gets interpreted as data!
- Sanitize user input: escape special characters
  - Single/double quotes, comment characters, etc.

SQL Injection Defense

- Make sure data gets interpreted as data!
- Sanitize user input: escape special characters
- Better approach: prepared/parametrized statements – declare what is data!

```
$pstmt = $db->prepare( "SELECT * FROM Orders
    WHERE Recipient = :name” );
$pstmt >bindParam(‘:name’, $user);
$pstmt->execute();
```

OS Command Injection

OS Command Injection

- Example vulnerable PHP code

  $file = $_POST[‘file’];

  system(“ls -l **$file** ”);

  User POST **foo; rm -rf /**

- Defense
  - Sanitize user input
  - Use filesize(), fileperms() etc. instead of system()

Compare Code Injection Attacks

- Control flow hijacking, cross-site scripting, SQL injection, OS command injection
  - All due to confusion between code and data
  - Account for 6 of top 10 software weaknesses
- Same ideas in defenses
  - Validate/sanitize user input vs. check length
  - Allowlist of trusted code
  - Unsafe vs. safe functions

2024 CWE Top 25 Most Dangerous Software Weaknesses

1. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
   CWE-79 | CVEs in KEV: 3 | Rank Last Year: 2 (up 1) ▲
2. Out-of-bounds Write
   CWE-787 | CVEs in KEV: 18 | Rank Last Year: 1 (down 1) ▼
3. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
   CWE-89 | CVEs in KEV: 4 | Rank Last Year: 3
4. Cross-Site Request Forgery (CSRF)
   CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9 (up 5) ▲
5. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
   CWE-22 | CVEs in KEV: 4 | Rank Last Year: 8 (up 3) ▲
6. Out-of-bounds Read
   CWE-125 | CVEs in KEV: 3 | Rank Last Year: 7 (up 1) ▲
7. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
   CWE-78 | CVEs in KEV: 5 | Rank Last Year: 5 (down 2) ▼
8. Use After Free
   CWE-416 | CVEs in KEV: 5 | Rank Last Year: 4 (down 4) ▼
9. Missing Authorization
   CWE-862 | CVEs in KEV: 0 | Rank Last Year: 11 (up 2) ▲
10. Unrestricted Upload of File with Dangerous Type
    CWE-434 | CVEs in KEV: 0 | Rank Last Year: 10

SQL Access Control

SQL Access Control

- Primarily uses Discretionary Access Control (DAC) and Role-based Access Control (RBAC)
  - db_owner, db_datawriter, db_datareader, …
- Owners can grant privileges on resources
  - Privileges include SELECT, INSERT, GRANT
  - Resources include databases, tables, rows

Example

```
CREATE TABLE Employee(
    name varchar(50),
    salary int,
    email varchar(50),
    manager varchar(50)
);

GRANT SELECT ON Employee TO Bob;
```

View-Based Access Control

- What if we don't want Bob to see salary?
- Views allow finer control of access to data
  - A view is not a partial copy, but a "filter" query

```
CREATE VIEW PubEmployee AS
    SELECT name, email FROM Employee;
GRANT SELECT ON PubEmployee TO Bob;
```

View-Based Access Control

- What if we want Bob to see salary of his own team members but not other employees?
- Views can be used for row-level access control

```
CREATE VIEW TeamBob AS
    SELECT * FROM Employee
    WHERE manager='Bob';
GRANT SELECT ON TeamBob TO Bob;
```

SQL Access Control and Injection

- Is it a good idea to use SQL access control to defend against SQL injection?

- Not in current convention where there is only one user of the database -- the web server

SQL Access Control and Injection

- Is it a good idea to use SQL access control to defend against SQL injection?

- Create a role with read-only access?
  - Help with injection of DROP, but not SELECT
- Every user has its own view?
  - May work but there are often cheaper solutions

Summary

- SQL injection: yet another code injection attack due to confusion between code and data
- Defenses: sanitize input, prepared statements
- Compare with other code injection attacks

- SQL access control: role-based and view-based