SMU SINGAPORE MANAGEMENT UNIVERSITY

School of Computing and Information Systems

CS440 Foundations of Cybersecurity

Web Security Part II

Recap Week 11

- HTTP is stateless
- Web applications establish sessions to maintain state
  - Session IDs are subject to hijacking attacks
- Same origin policy: isolate scripts and resources based on their origin
- Cross-Site Scripting: a kind of code injection attack
  - Can steal Session IDs or other information from client side
  - Three types: Stored XSS, Reflected XSS & Dom-based XSS
  - Defence strategies: filtering, improved access control

Overview

- Content
  - Cross Site Request Forgery (CSRF)
  - SQL injection (SQLi)

- After this module, you should be able to
    - Describe CSRF attacks and how to prevent these attacks
    - Describe SQL injection attacks and how to prevent them

Cross Site Request Forgery (CSRF)

Cross-site Request Forgery (CSRF)

- Parties involved: attacker, user, server
- Cross-site request forgery (XSRF, CSRF) exploits ‘trust’ a target server has in a user to execute actions at the server with the user’s privileges
  - Trust: user is in some way authenticated at the server (cookie, authenticated SSL session, ...)
- Violates server’s assumption about the end point of a request

CSRF – Attack Pattern

- User visits a page at attacker's site and clicks a link;
- Or user clicks on a link in email, social media, etc.

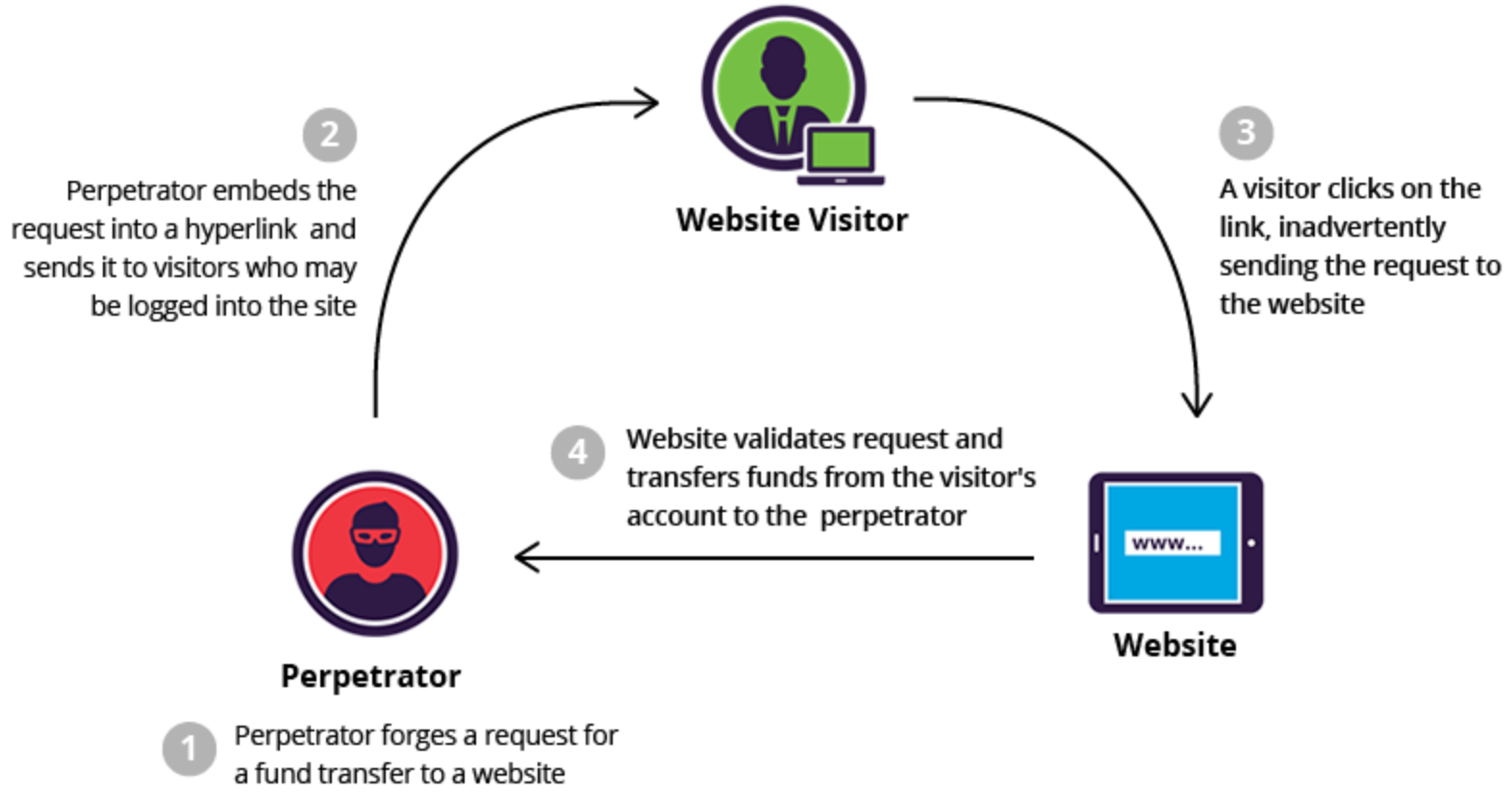
- When the user clicks the link (crafted by the attacker), the user's browser will send a request to the target server (pointed by the link)
- Server authenticates request as coming from user, because it comes from the user's browser
  - Actions (name and value pairs) in the request are executed by server with access rights of the authenticated user
  - Session hijacking attack

CSRF Example

- A typical GET request for a $100 bank transfer:
- GET http://bank.com/transfer.php?acct=User&amount=$100
- An attack crafts a malicious URL:
- http://bank.com/transfer.php?acct=Attacker&amount=$100

- He can embed the URL in his website or distribute the link via social media
- Those who click the link while logged into their bank account will unintentionally transfer $100 to attacker

CSRF Example



CSRF Defenses

- Ultimate cause of attack: server only authenticates ‘the last hop’ of the entire request, but not the true origin of all its parts

- Defense
  - Authenticate requests (actions) at the level of the web application (‘above’ the browser)
  - Avoid GET requests whenever possible

CSRF Prevention Tokens

- Server creates a random “challenge” token associated with a user session/request
  - Per-request tokens are more secure than per-session tokens

- Tokens inserted into HTML forms

```
<form action=“/transfer.do” method=“post”>
   <input type="hidden" name=“CSRFToken”
      value=“OWY...E1ZMGYwMGEwOA==”>
...
</form>
```

CSRF Prevention Tokens

- Token value must be fresh
- Tokens must be unique and unpredictable

- Web application must validate each request by comparing received token to stored one
- Requests without token or with wrong token are discarded

CSRF Protection from User's Perspective

- Log off web applications when not in use
- Avoid simultaneously browsing while logged into a sensitive application (e.g. banking, transaction applications)
- You do not want them to "remember me"

SQL Injection

What is SQL?

- Most websites use a database to store data
  - such as usernames, passwords, etc.
- Web application reads, updates and inserts data in the database
- Interaction with the database done via SQL (Structured Query Language)

SQL

- SQL: Standard language for accessing relational databases
- Fetch a set of records

```
SELECT * FROM user WHERE ID='12'
```

- Add data to the table

```
INSERT INTO user (name,id) VALUES ('Jack',10)
```

- Modify data

```
UPDATE user SET name='Mike' WHERE id='12'
```

- Query syntax (mostly) independent of vendor

No need to write code in the exam

What is SQL Injection?

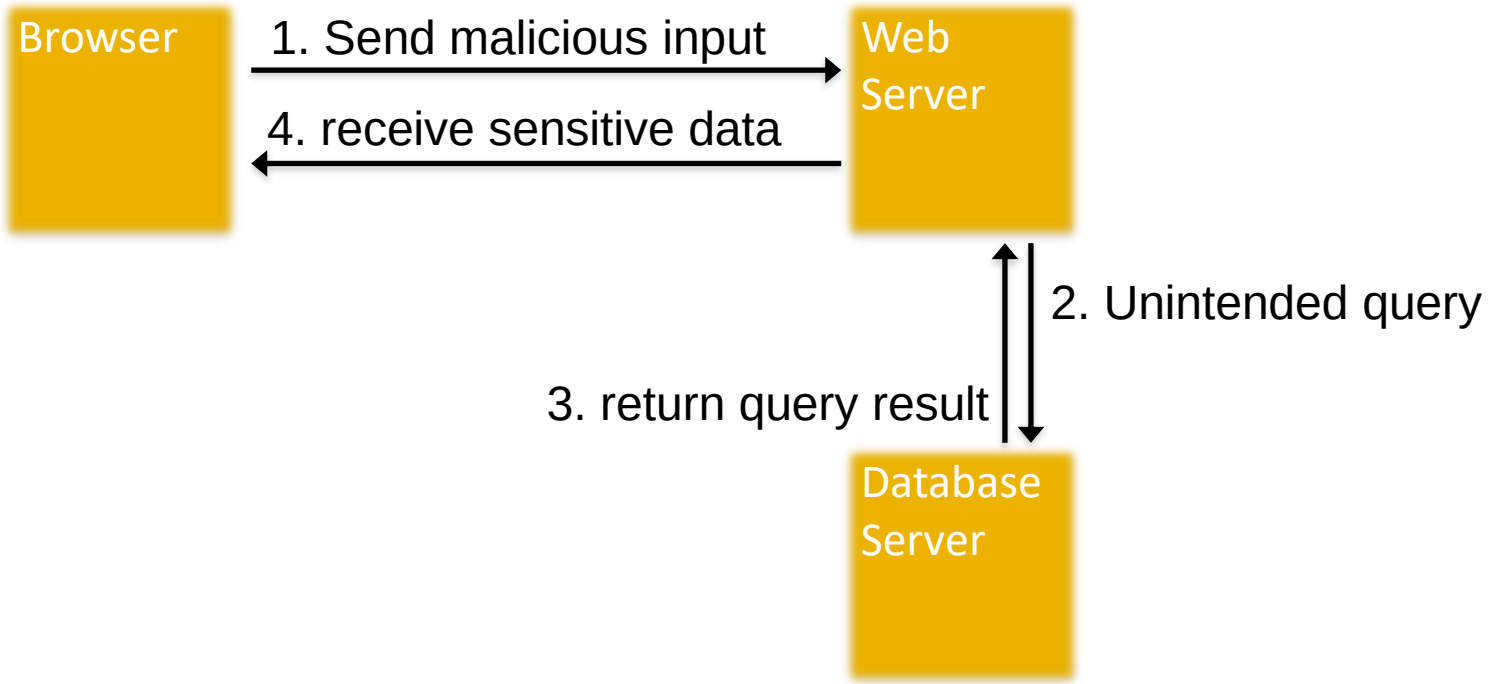
- Gain access to the database by manipulating user inputs used in SQL queries
  - Inject ***special characters*** ***(syntactic meaning to SQL parser/interpreter)*** such as ' # OR ; =
  - At places such as login forms and URLs:

    http://target.com/login.php?user=maliciousinput&psw=maliciousinputs

- Can obtain sensitive information such as admin passwords for further exploitation of the system

- Can be used to delete database tables, upload files, create reverse shell, etc.

Flow of SQL Injection Attack



Example: SQL Injection

User gets authenticated

Name	Surname	
John	Smith	…

Username John

Password 1234

OK

Web form

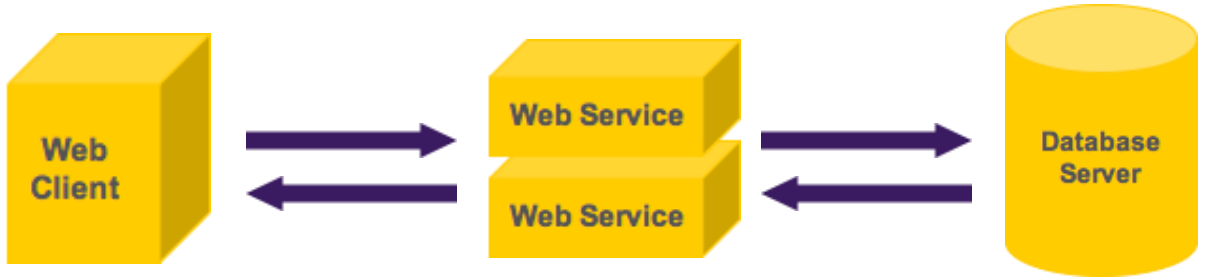
```
SELECT * FROM Users
WHERE
usr = 'John' AND
psw = '1234'
```

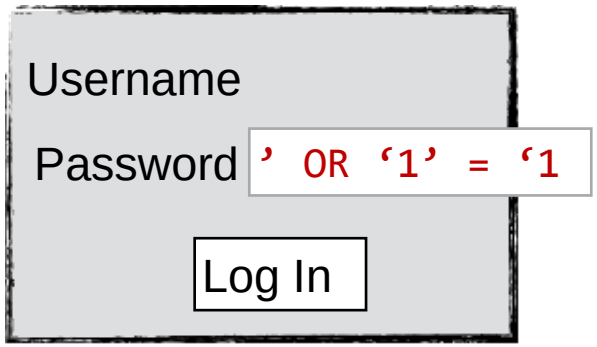
SQL query

Name	Surname	
John	Smith	…

Result

Example: SQL Injection





Web form

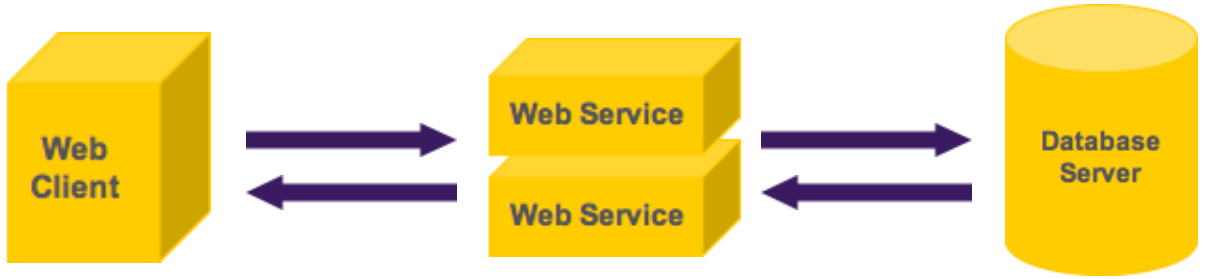
```
SELECT * FROM Users
WHERE
usr = '' AND psw = ''
OR  '1'='1'
```

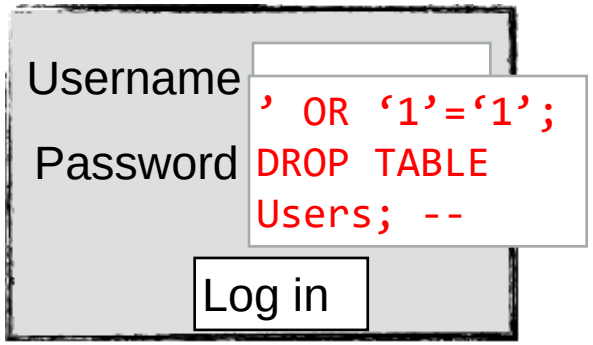
SQL query

Name	Surname	…
Aalto	Aron	…
John	Smith	…
…	…	…

Query result

Example: SQL Injection



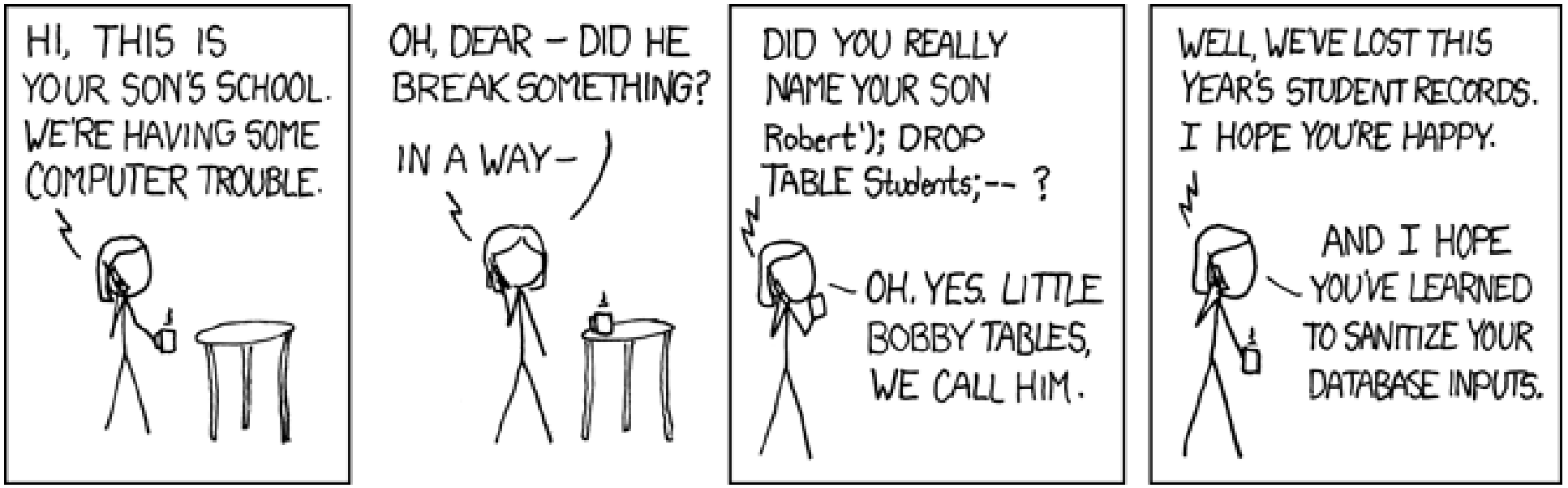



```
SELECT * FROM Users
WHERE
usr = '' AND psw = ''
OR  '1'='1'; DROP TABLE
Users; --
```

Name	Surname	...

Eliminate all user accounts

Example: SQL Injection



http://xkcd.com

Remark on SQL Injection

- Single quote ’ is the string delimiter in SQL
  - E.g. name=‘Jack’ AND age=11
- In a web application, a server-side script (e.g. written in PHP) may construct SQL query as a string from user input
- String passed to DBMS and executed as SQL query
- User input can now be interpreted as code by DBMS!
- Broken abstraction:
  - No clear distinction between data and code

Defence against SQL Injection

- Two fundamental defences:
  1. Sanitize or validate user inputs so that dangerous inputs are made innocuous
     a. Filtering (blacklist: [', ", \, …], whitelist: ['Mary', 'Jane', 'Jack', …])
     b. Escaping - Replace dangerous characters with encodings – e.g. ' -> \'
     (\ -> tells the SQL interpreter not to interpret the subsequent character with any special meaning)
  2. Bound parameters so that user inputs cannot be mistaken for code (clean solution to the problem)
- Additional precautions:
  - Avoid verbose error messages; error messages for invalid inputs can leak information about database, relations, names of columns, etc.
  - Least privilege: do not connect as sysadmin

Example: Bound parameters with prepared statements

- Pre-compile query with placeholders; execute query with actual user input:

```
$uname = $_GET['username'];
 $pwd = $_GET['password'];
 // :xxxx serves as placeholders in prepared statement
$sql = "SELECT username FROM users WHERE username = :uname AND
                                  password = :pwd";
// Replace :xxxx in the statement with actual input
 $stmt = $pdo->prepare($sql);
 $stmt->setFetchMode(PDO::FETCH_ASSOC);
$stmt->bind_param(':uname', $uname, PDO::PARAM_STR);
 $stmt->bindParam(':pwd', $pwd, PDO::PARAM_INT);

// Run the query
$stmt->execute();
```

placeholders

No need to write code in the exam

Take Away

- Cross Site Request Forgery is a session hijacking attack. It confuses the server because the request comes from the user who is authenticated to a session.
- SQL queries are used by web to interact with databases
- SQL injection: Attack on the database through the application
  - User inputs are used by application in constructing the queries
  - Attack exploits this to gain unauthorized access to data stored in database