Web Security

SQL Injection, XSS, CSRF, Parameter Tampering,
Session Hijacking

greenwich.edu.vn

UNIVERSITY of GREENWICH
Alliance with FPT Education

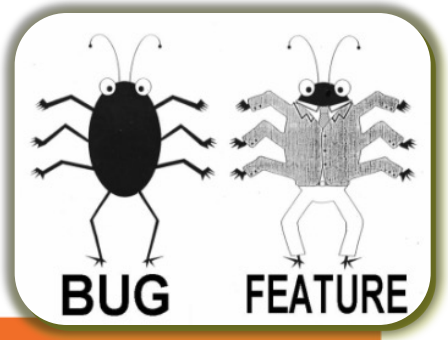
Table of Contents

WEB SECURITY MAIN CONCEPTS

UNIVERSITY of GREENWICH
Alliance with FPT Education

Feature or Bug

- **Is Software Security a Feature**
  - Most people consider software security as a necessary feature of a product
- **Is Security Vulnerability a Bug?**
  - If the software "failed" and allowed a hacker to see personal info, most users would consider that a software bug



UNIVERSITY of GREENWICH
Alliance with FPT Education

Reasons for Failures

- Software failures usually happen **spontaneously**
  - Without intentional mischief
- Failures can be result of **malicious attacks**
  - For the Challenge/Prestige
  - Curiosity driven
  - Aiming to use resources
  - Vandalizing
  - Stealing

UNIVERSITY of GREENWICH
Alliance with FPT Education

Golden Rules!

- Maximum Simplicity
  - More complicated – greater chance for mistakes
- Secure the Weakest Link
  - Hackers attack where the weakest link is
- Limit the Publicly Available Resources
- Incorrect Until Proven Correct
  - Consider each user input as **incorrect**
- The Principle of the "Weakest Privilege"
- Security in Errors (Remain stable)
- Provide Constant Defense (also use backups)

SQL INJECTION

What is SQL Injection and How to Prevent It?

What is SQL Injection?

```
$loginQuery = "SELECT * FROM users
    WHERE username='{$_POST['user']}' AND
          password='{$_POST['pass']}'";
$result = mysql_query($loginQuery);
```

- Try the following queries:
  - `'` → crashes
  - `' or ''='` → Login with any user
  - `'; INSERT INTO Messages(MessageText, MessageDate) VALUES ('Hacked!!!', '1.1.1980')--` → injects a message

UNIVERSITY of GREENWICH
Alliance with FPT Education

How Does SQL Injection Work?

- The following SQL commands are executed:
  - Usual search (no SQL injection):

```
SELECT * FROM Messages WHERE MessageText LIKE '%nakov%'"
```

  - SQL-injected search (matches all records):

```
SELECT * FROM Messages WHERE MessageText LIKE '%%%'"
```

```
SELECT * FROM Messages WHERE MessageText LIKE '%' or 1=1 --%'"
```

  - SQL-injected INSERT command:

```
SELECT * FROM Messages WHERE MessageText
LIKE '%'; INSERT INTO Messages(MessageText, MessageDate)
VALUES ('Hacked!!!', '1.1.1980') --%'"
```

Another SQL Injection Example

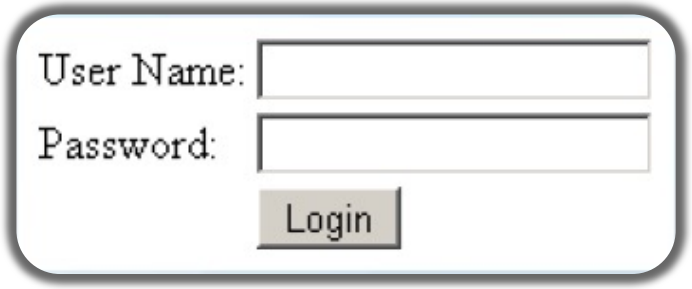
- Original SQL Query:

```
String sqlQuery = "SELECT * FROM user WHERE name = '" +
username + "' AND pass='" + password + "'"
```

- Setting username to **John** & password to **' OR '1'= '1** produces

```
String sqlQuery = SELECT * FROM user WHERE name =
'Admin' AND pass='' OR '1'='1'
```

- Result: If a user Admin exists – he is logged in without password



UNIVERSITY of GREENWICH
Alliance with FPT Education

Preventing SQL Injection

- Ways to prevent the SQL injection:
  - **SQL-escape** all data coming from the user:
    - Not recommended: use as last resort only!
  - Preferred approach:
    - Use **ORM**
    - Use **parameterized queries**

CROSS SITE SCRIPTING (XSS)

What is XSS and How to Prevent It?

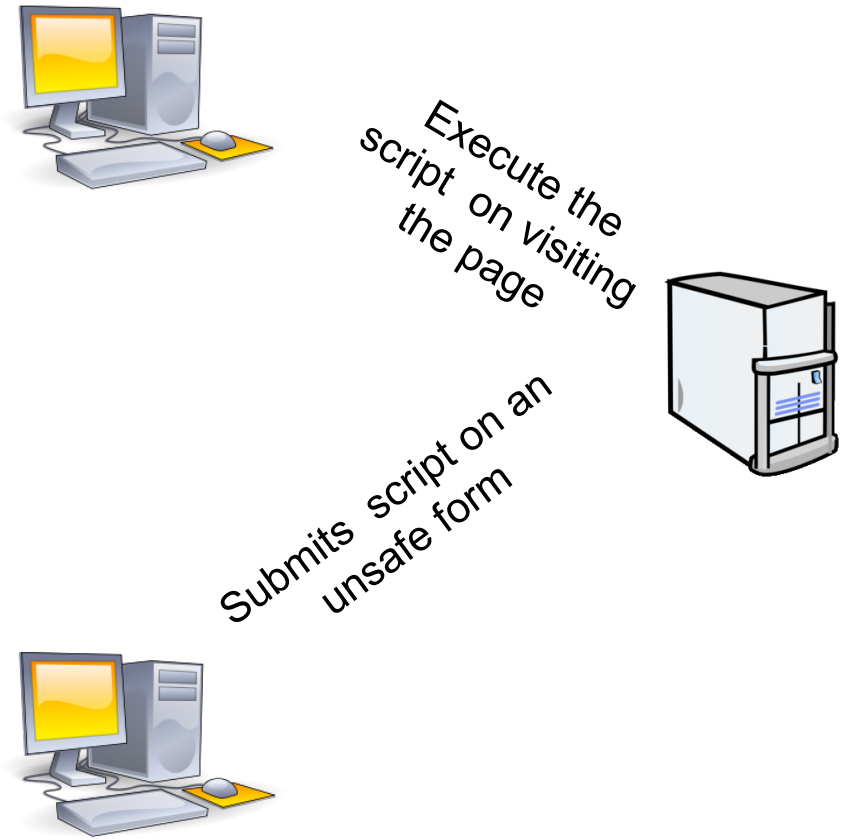
UNIVERSITY of GREENWICH
Alliance with FPT Education

XSS Attack

- **C**ross-**S**ite **S**cripting (**XSS**) is a common security vulnerability in Web applications
  - Web application is let to display a JavaScript code that is executed at the client's browser
    - Crackers could take control over sessions, cookies, passwords, and other private data
- How to prevent from XSS?
  - Validate the user input
  - Perform HTML escaping when displaying text data in a Web control

XSS

- Cross-site scripting attack
  - Cookie theft
  - Account hijacking
  - Modify content
  - Modify user settings
  - Download malware
  - Submit CRSF attack
  - Password prompt



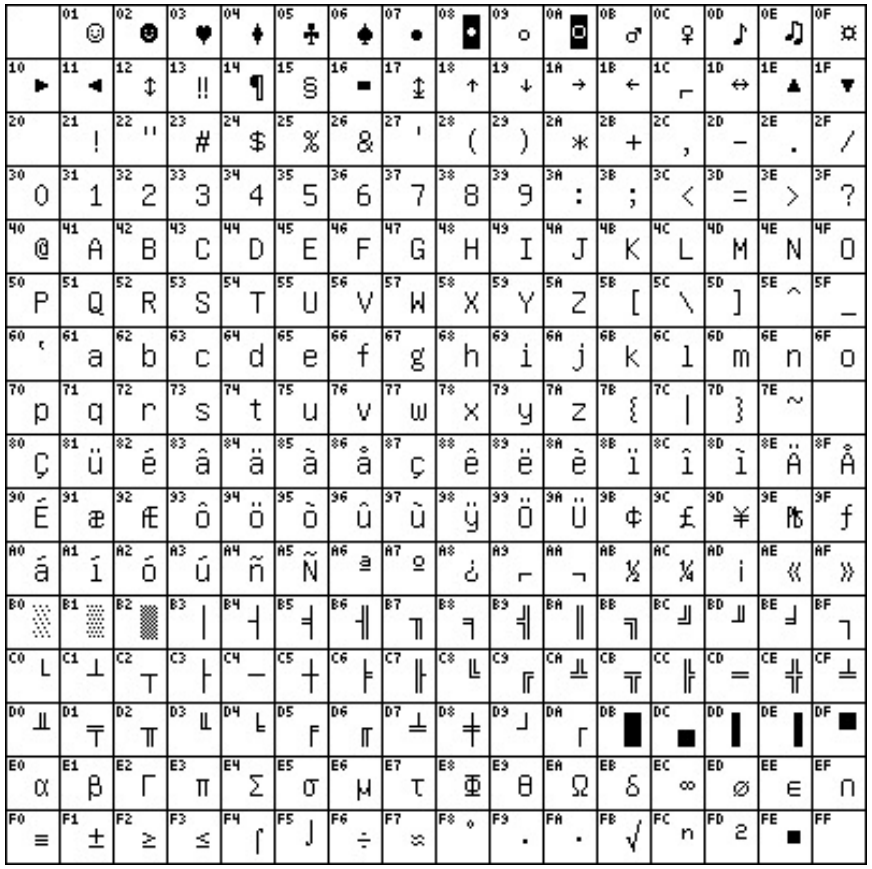
UNIVERSITY of GREENWICH
Alliance with FPT Education

What is HTML Escaping?

- **HTML escaping** is the act of replacing special characters with
their HTML entities
  - Escaped characters are interpreted as character data instead of mark up
- Typical characters to escape
  - <, > – start / end of HTML tag
  - & – start of character entity reference
  - ', " – text in single / double quotes

HTML Character Escaping

- Each character could be presented as **HTML entity** escaping sequence
- Numeric character references:
  - 'λ' is `&#955;`, `&#x03BB;` or `&#X03bb;`
- Named HTML entities:
  - 'λ' is `&lambda;`
  - '<' is `<`
  - '>' is `>`
  - '&' is `&`
  - " (double quote) is `&quot;`



UNIVERSITY of GREENWICH
Alliance with FPT Education

How to Encode HTML Entities?

- HTML encodes a string and returns the encoded (html-safe) string
  Example (in PHP):

```
echo htmlspecialchars("The image tag: <img>");
```

```
echo htmlentities("The image tag: <img>");
```

- HTML Output:

```
The image tag: &lt;img&gt;
```

- Web browser renders the following:

```
The image tag: <img>
```

CROSS-SITE REQUEST FORGERY

What is CSRF and How to Prevent It?

UNIVERSITY of GREENWICH
Alliance with FPT Education

What is CSRF?

- **C**ross-**S**ite **R**equest **F**orgery (**CSRF / XSRF**) is a web security attack
  over the HTTP protocol
  - Allows **executing unauthorized commands** on behalf of some authenticated user
    - E.g. to transfer some money in a bank system
  - The user has valid permissions to execute the requested command
  - The attacker uses these permissions to send a forged HTTP request unbeknownst to the user
    - Through a link / site / web form that the user is allured to open

UNIVERSITY of GREENWICH
Alliance with FPT Education

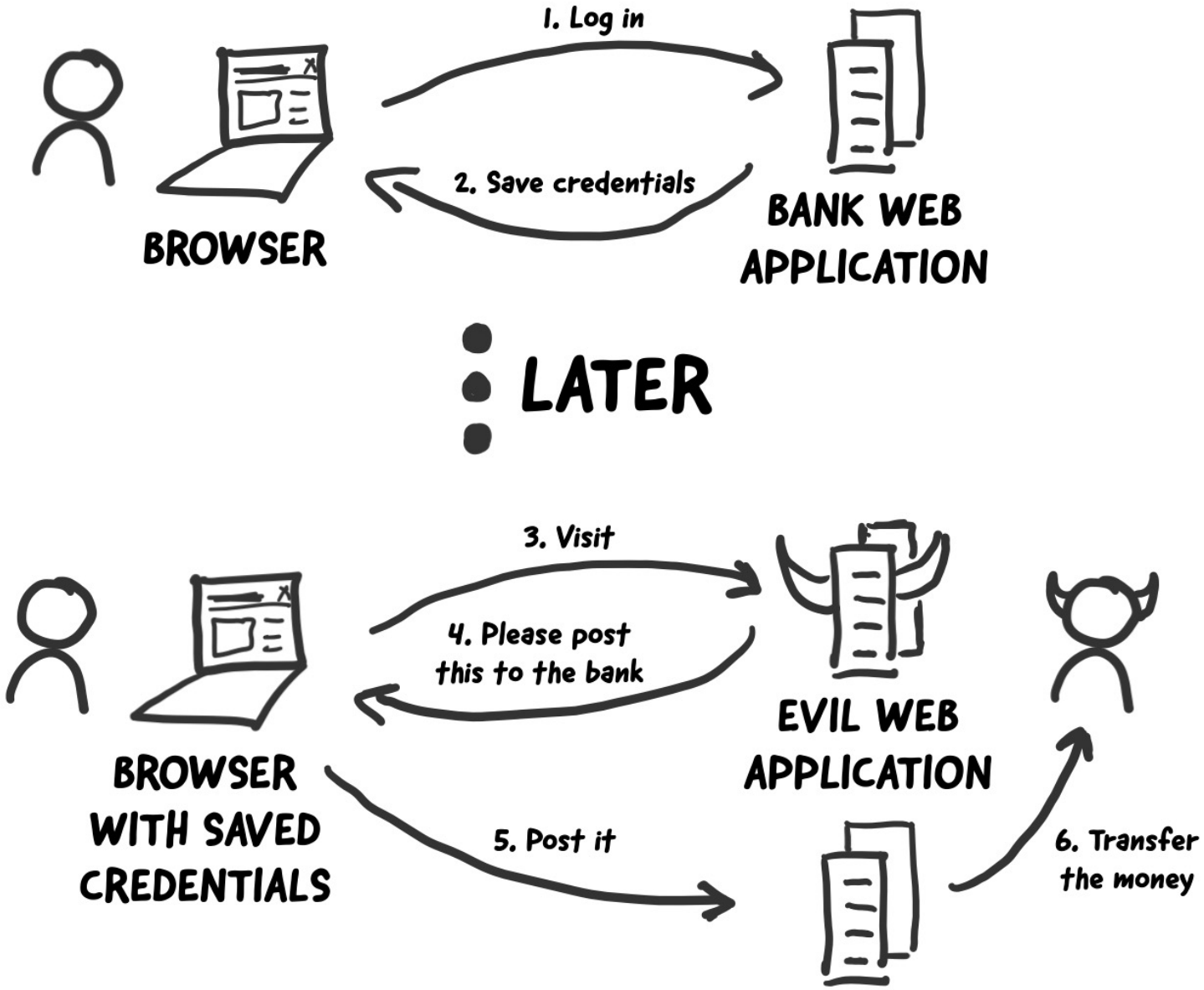
CSRF Explained

- The user has a valid authentication cookie for the site victim.org (remembered in the browser)
- The attacker asks the user to visit some evil site, e.g. http://evilsite.com
- The evil site sends HTTP GET / POST to victim.org and does something
  evil
  - Through a JavaScript AJAX request
  - Using the browser's authentication cookie
- The victim.org performs the unauthorized command on behalf of the authenticated user

UNIVERSITY of GREENWICH
Alliance with FPT Education

CSRF

- Cross-site request forgery attack



UNIVERSITY of GREENWICH
Alliance with FPT Education

Prevent CSRF in PHP

- To prevent **CSRF** attacks in PHP apps use random generated tokens
  - Put hidden field with random generated token in the HTML forms:

```
$_SESSION['formToken'] = uniqid(mt_rand(), true);
```

```
<form action="" method="POST">
  <input type="text" name="message" />
  <input type="hidden" name="formToken" value="$_SESSION['formToken']" />
</form>
```

  - Verify anti-CSRF token in each controller action that should be protected:

```
if (!isset($_POST['formToken']) ||
    $_POST['formToken'] != $_SESSION['formToken']) {
  throw new Exception('Invalid request!');
  exit; }
```

PARAMETER TAMPERING

What is Parameter Tampering and How to Prevent It?

UNIVERSITY of GREENWICH
Alliance with FPT Education

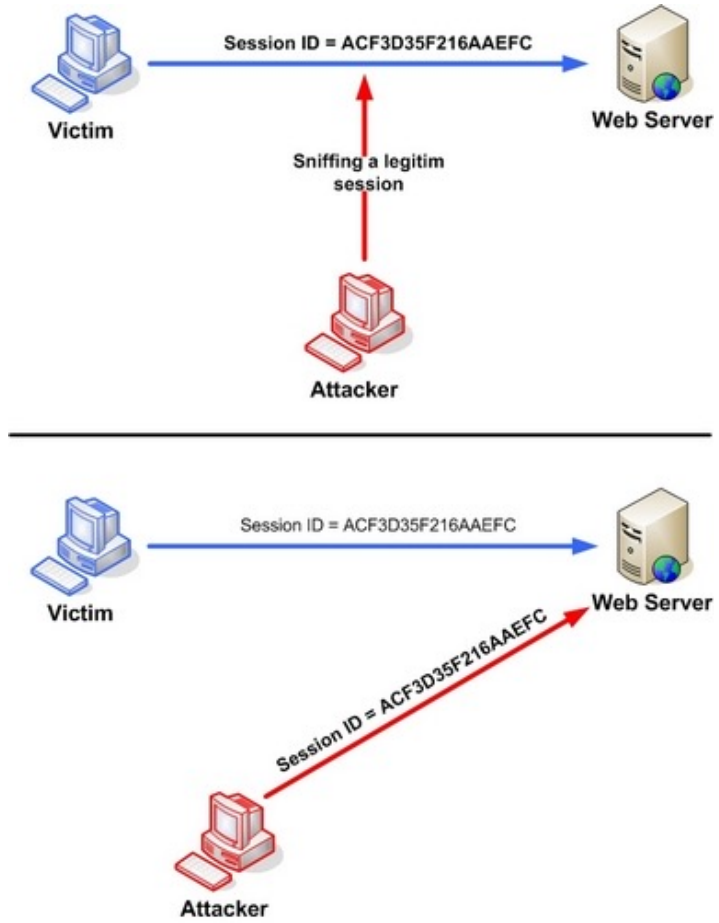
What is Parameter Tampering?

- What is Parameter Tampering?
  - Malicious user alters the HTTP request parameters in unexpected way
  - Altered **query string** (in GET requests)
  - Altered **request body** (form fields in POST requests)
  - Altered **cookies** (e.g. authentication cookie)
  - **Skipped data validation** at the client-side
  - **Injected parameter** in MVC apps

SESSION HIJACKING

Session Hijacking

- Capture a valid token session using a sniffer
- Use the valid session token to gain unauthorized access to the server
- Always use SSL when sending sensitive data!
- You should use Man in the Middle attack to sniff the session token



Other Threats

- Semantic URL attacks
  - URL Manipulation
- Man in the Middle (MiTM)
- Brute force (use CAPTCHA!)
- Insufficient Access Control
- Error messages can reveal information
- Phishing
- Security flows in other software you are using
- Social Engineering