# ENEE 459-C
# Computer Security

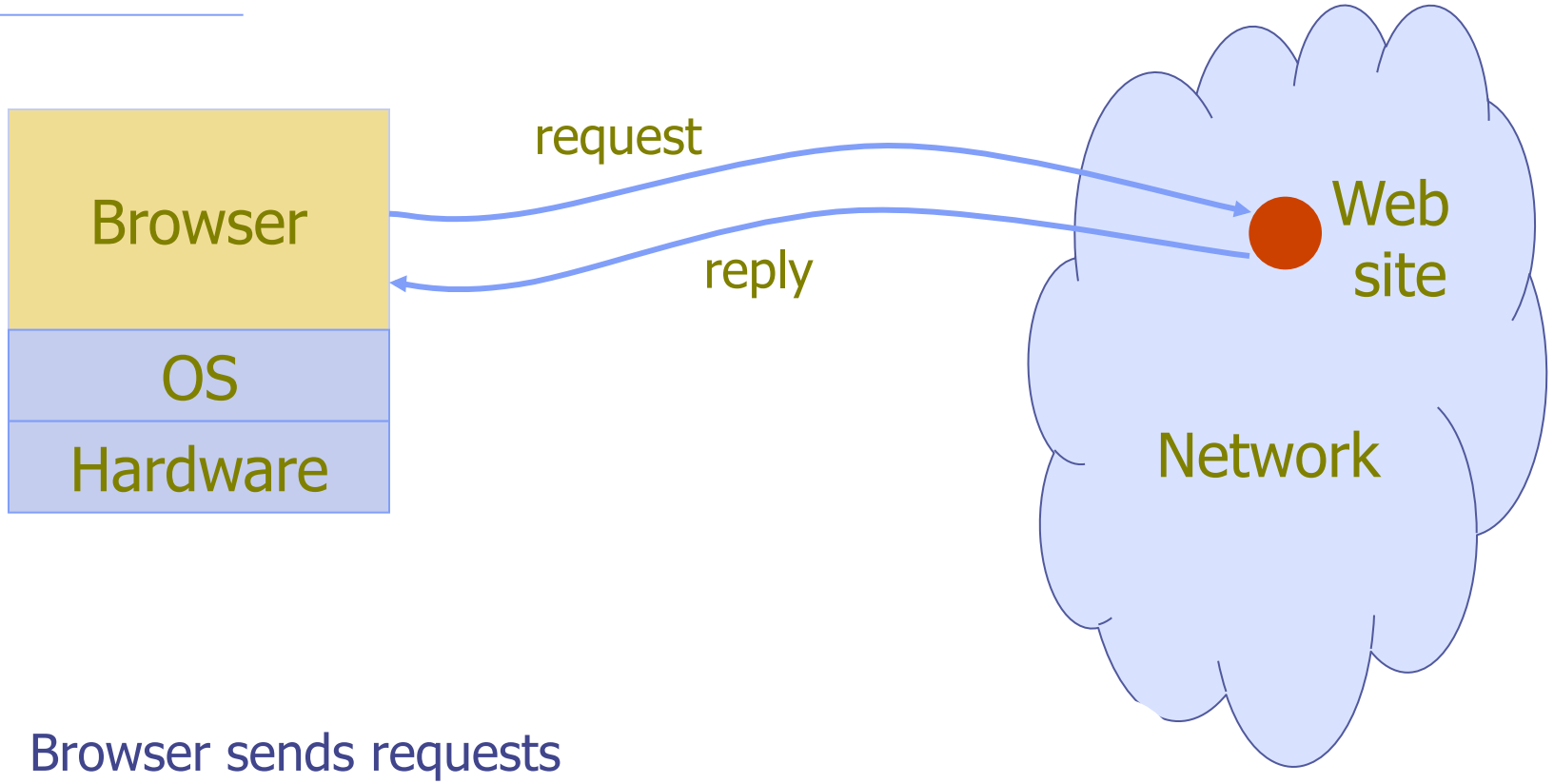
## Web Security

UNIVERSITY OF
MARYLAND

# Web, everywhere

- Many tasks are done through web
  - Online banking, online shopping
  - Database access
  - System administration
- Web applications and web users are targets of many attacks
  - Information leakage
  - Cross site scripting
  - SQL injection

# Web Browser and Network




- Browser sends requests
- Web site sends response pages, which may include code
- Interaction susceptible to network attacks

# Web Security Issues

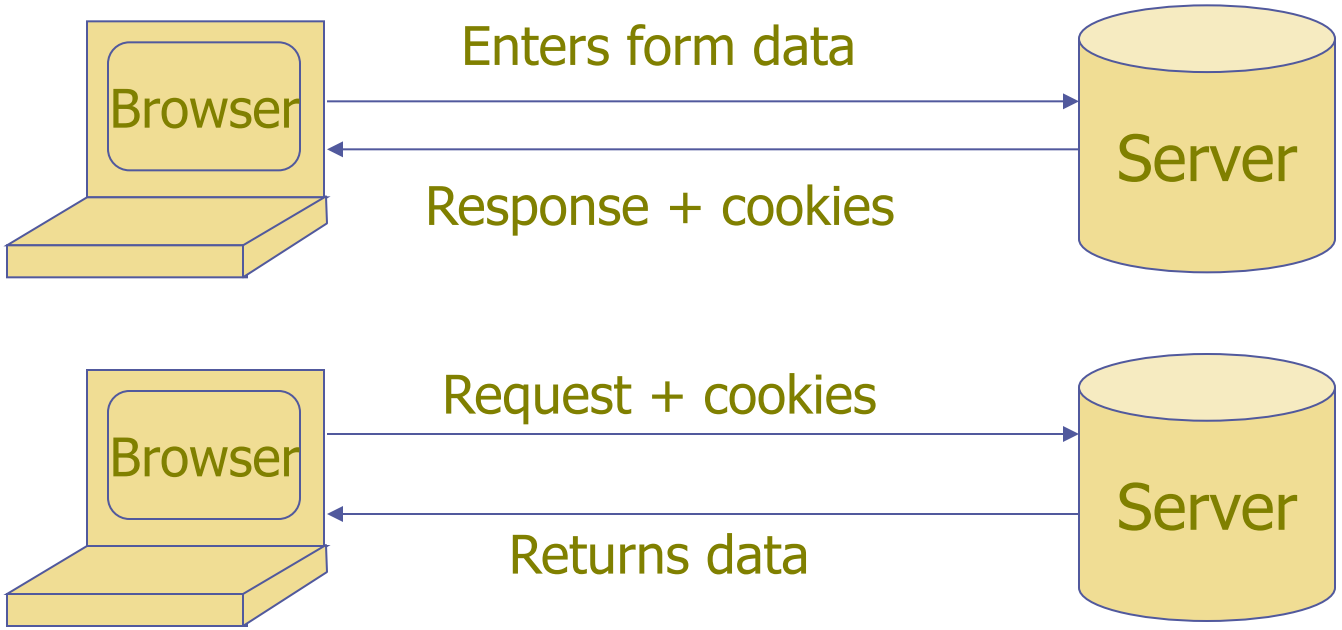
- Secure communications between client & server
  - HTTPS (HTTP over SSL)
- User authentication & session management
  - cookies & other methods
- Web application security
  - program analysis
- Web site authentication (e.g., anti-phishing)
  - certificates

# HTTP: HyperText Transfer Protocol

- Browser sends HTTP requests to the server
  - Methods: GET, POST, HEAD, …
  - GET: to retrieve a resource (html, image, script, css,…)
  - POST: to submit a form (login, register, …)
  - HEAD: to retrieve only metadata
- Server replies with a HTTP response
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

# Use Cookies to Store State Info

- Cookies
  - A cookie is a piece of information created by a website to store information on your computer




Http is stateless protocol; cookies add state

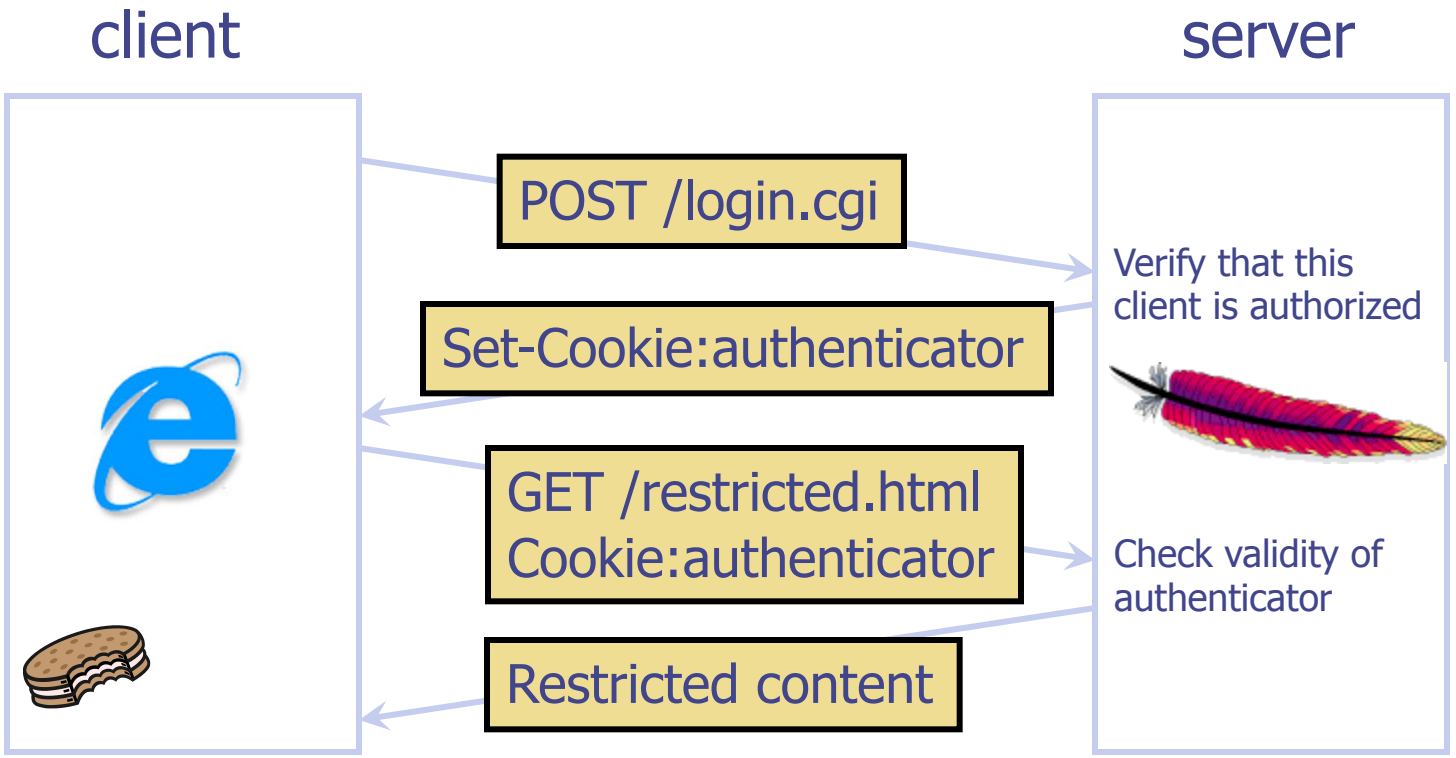
# Cookies

- Stored by the browser at the client
- Used by the web applications
  - used for authenticating, tracking, and maintaining specific information about users
    - e.g., site preferences, contents of shopping carts
  - data may be sensitive
  - may be used to gather information about specific users

# Web Authentication via Cookies

- HTTP is stateless
  - How does the server recognize a user who has signed in?
- Servers can use cookies to store state on client
  - After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
    - Client cannot forge authenticator on his own (session id)
  - With each request, browser presents the cookie
  - Server verifies the authenticator

# A Typical Session with Cookies




Authenticators must be unforgeable and tamper-proof

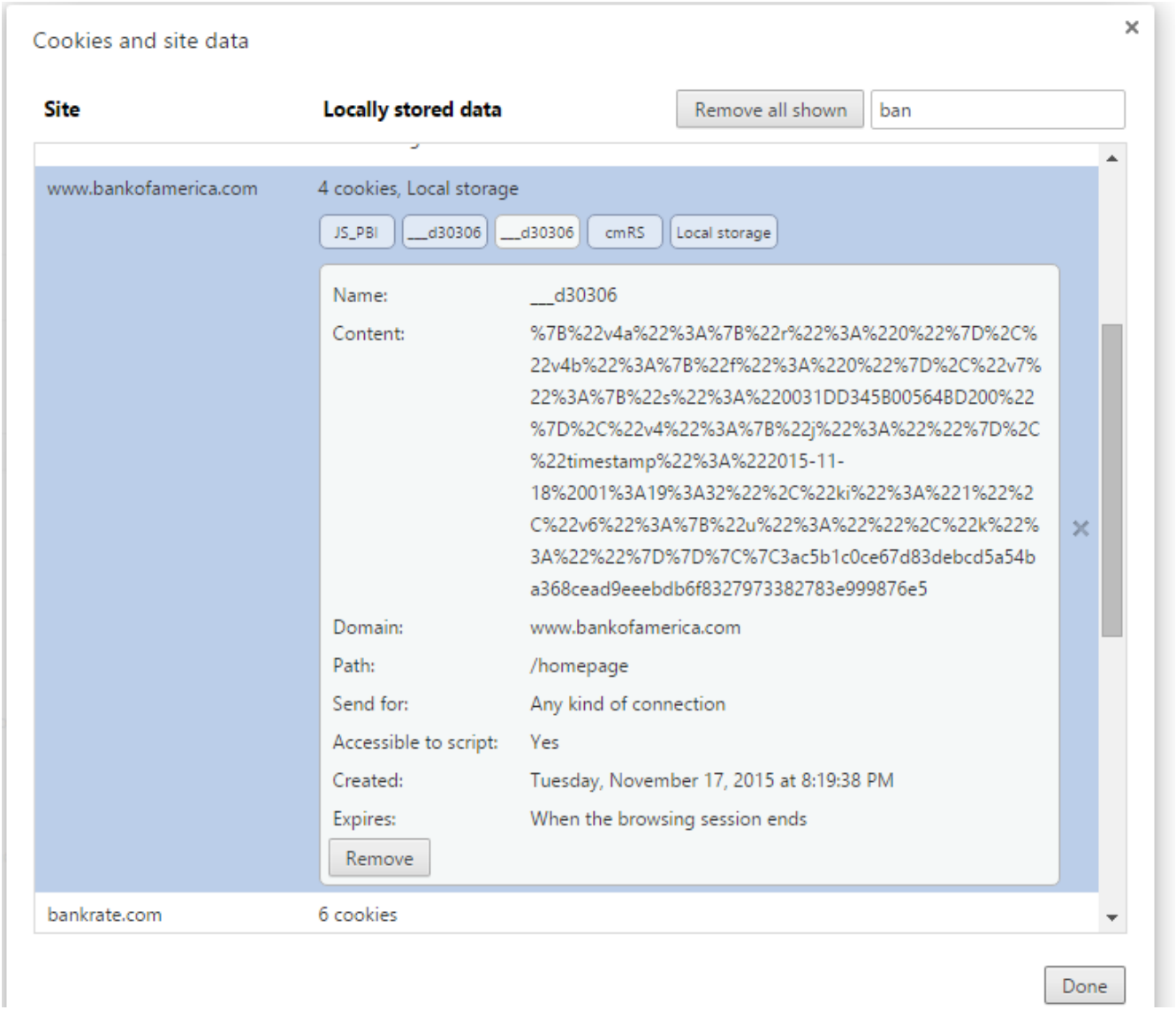
(malicious clients shouldn't be able to modify an existing authenticator)

How to design it?

# Authentication cookies

- cookie(userID)=
- (userID||time, MAC(K,userID||time))
- time: time the cookie was created
- K:secret key known by server only
- The server needs to make sure that
  - Current_time – time  <= cookie_duration
- If logged out or changed password before expiration date, then previous cookie gets invalidated

# Cookies Fields



# Cross Site Scripting

# Client Side Scripting

- Web pages (HTML) can embed dynamic contents (code) that can execute on the browser

- JavaScript
  - embedded in web pages and executed inside browser
- VBScript
  - similar to JavaScript, only for Windows
- Java applets
  - small pieces of Java bytecodes that execute in browsers

# HTML and Scripting

```
<html>
<script>
      var num1, num2, sum
      num1 = prompt("Enter first number")
      num2 = prompt("Enter second number")
      sum = parseInt(num1) + parseInt(num2)
      alert("Sum = " + sum)
</script>
   </html>
```

Browser receives content, displays HTML and executes scripts

# Scripts are Powerful

- Client-side scripting is powerful and flexible, and can access the following resources
  - Local files on the client-side host
    - read / write local files
  - Webpage resources maintained by the browser
    - Cookies

# Browser as an Operating System

- Web users visit multiple websites simultaneously
- A browser serves web pages (which may contain programs) from different web domains
  - i.e., a browser runs programs provided by mutually untrusted entities
  - Running code one does not know/trust is dangerous
  - A browser also maintains resources created/updated by web domains
- Browser must confine (sandbox) these scripts so that they cannot access arbitrary local resources
- Browser must have a security policy to manage/protect browser-maintained resources and to provide separation among mutually untrusted scripts

# Same Origin Policy

- The basic security model enforced in the browser
- SoP isolates the scripts and resources downloaded from different origins
  - E.g., evil.org scripts cannot access bank.com resources
- Use origin as the security principal
- Origin = domain name + protocol + port
  - all three must be equal for origin to be considered the same

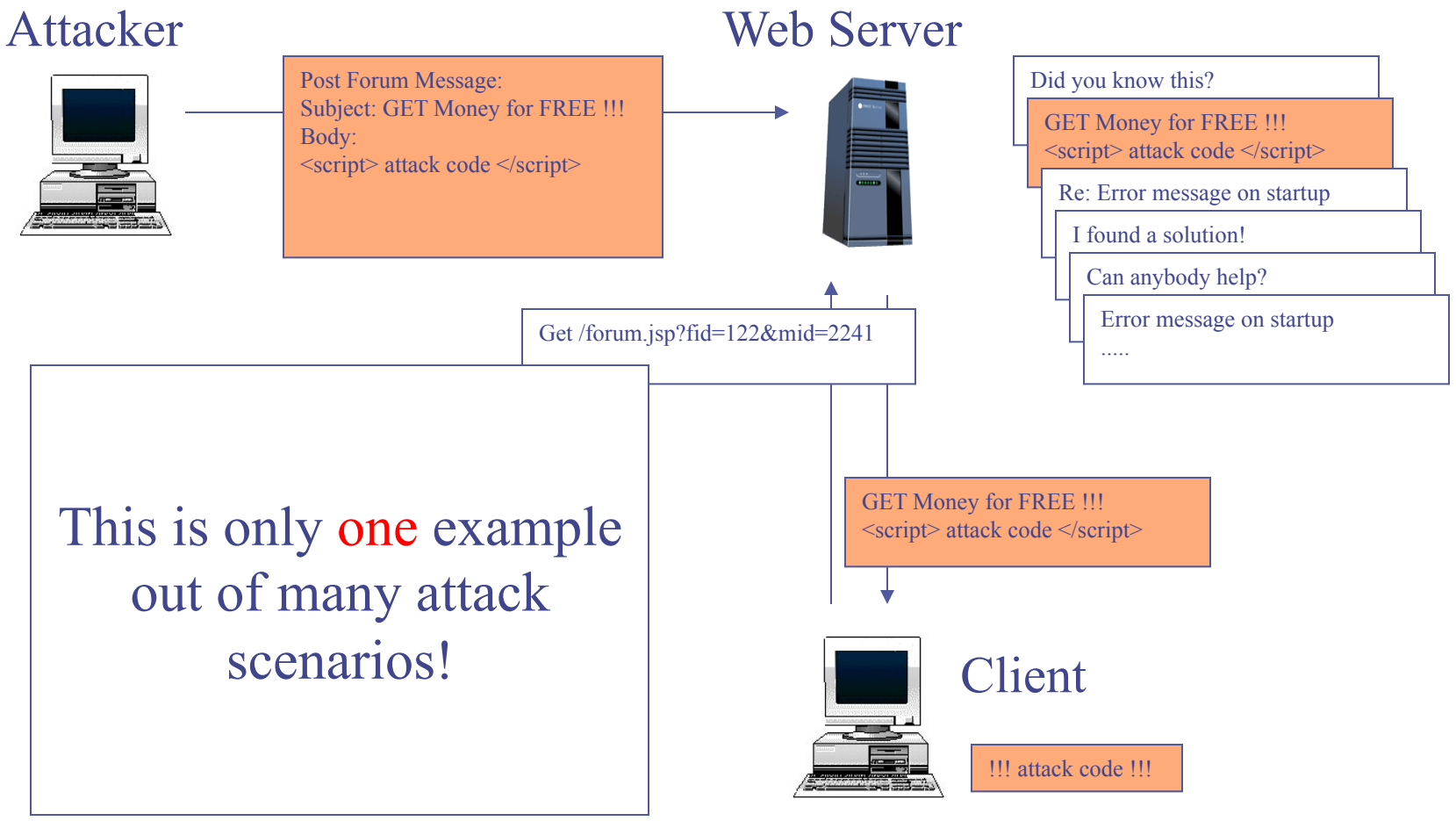
# Problems with S-O Policy

- Poorly enforced on some browsers
  - Particularly older browsers
- Limitations if site hosts unrelated pages
  - Example: Web server often hosts sites for unrelated parties
    - http://www.example.com/account/
    - http://www.example.com/otheraccount/
  - Same-origin policy allows script on one page to access properties of document from another
- Can be bypassed in Cross-Site-Scripting attacks
- Usability: Sometimes prevents desirable cross-origin resource sharing

# Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - scripts controlled by the same-origin policy
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage (these inputs can have scripts)

# XSS-Attack: General Overview



# How XSS Works on Online Blog

- Everyone can post comments, which will be displayed to everyone who view the post
- Attacker posts a malicious comment that includes scripts (which reads local authentication credentials and sends to the attacker)
- Anyone who view the post can have local authentication cookies stolen
- Web apps  will check that posts do not include scripts, but the check sometimes fail.
- Bug in the web application. Attack happens in browser.

# XSS Example

- http://www.steve.org.uk/Security/XSS/Tutorial/simple.html

# Protection against XSS attacks

- Sanitize the input
- Make sure it does not contain any scripts!

# SQL Injection Attack

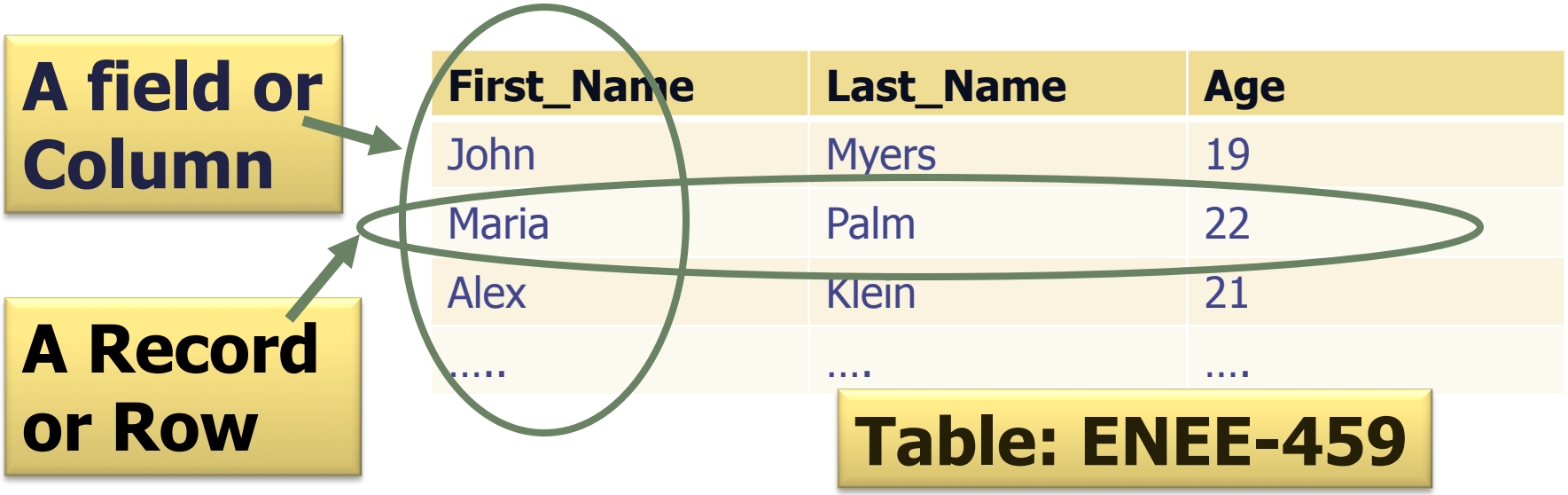
- Many web applications take user input from a form
- Often this user input is used literally in the construction of a SQL query submitted to a database. For example:

```
SELECT user FROM table
    WHERE name = 'user_input';
```

- An SQL injection attack involves placing SQL statements in the user input

# SQL: Standard Query Language

- SQL lets you access and manage (Query) databases
- A database is a large collection of data organized in tables for rapid search and retrieval, with fields and columns



# SQL Syntax

```
SELECT First_Name
FROM ENEE-459
WHERE age=21
```

- SELECT statement is used to select data FROM one or more tables in a database
- Result-set is stored in a result table
- WHERE clause is used to filter records

# SQL Syntax

```
SELECT  Last_Name
FROM ENEE-459
WHERE age=21
ORDER BY First_Name ASC
LIMIT 3
```

- ORDER BY is used to order data following one or more fields (columns)
- LIMIT  allows to retrieve just a certain numbers of records (rows)

# Login Authentication Query

- Standard query to authenticate users:

  select * from users where user='$usern' AND pwd='$password'

- Classic SQL injection attacks
  - Server side code sets variables $username and $passwd from user input to web form
  - Variables passed to SQL query

  select * from users where user='$username' AND pwd='$passwd'

- Special strings can be entered by attacker

  select * from users where user=M' OR '1=1 AND pwd=M' OR '1=1

- Result: access obtained without password
- Solution: Careful with single quote characters
- Filter them out!