CSC429 – Computer Security

LECTURE 9
WEB SECURITY

Mohammed H. Almeshekah, PhD
meshekah@ksu.edu.sa

Web Security

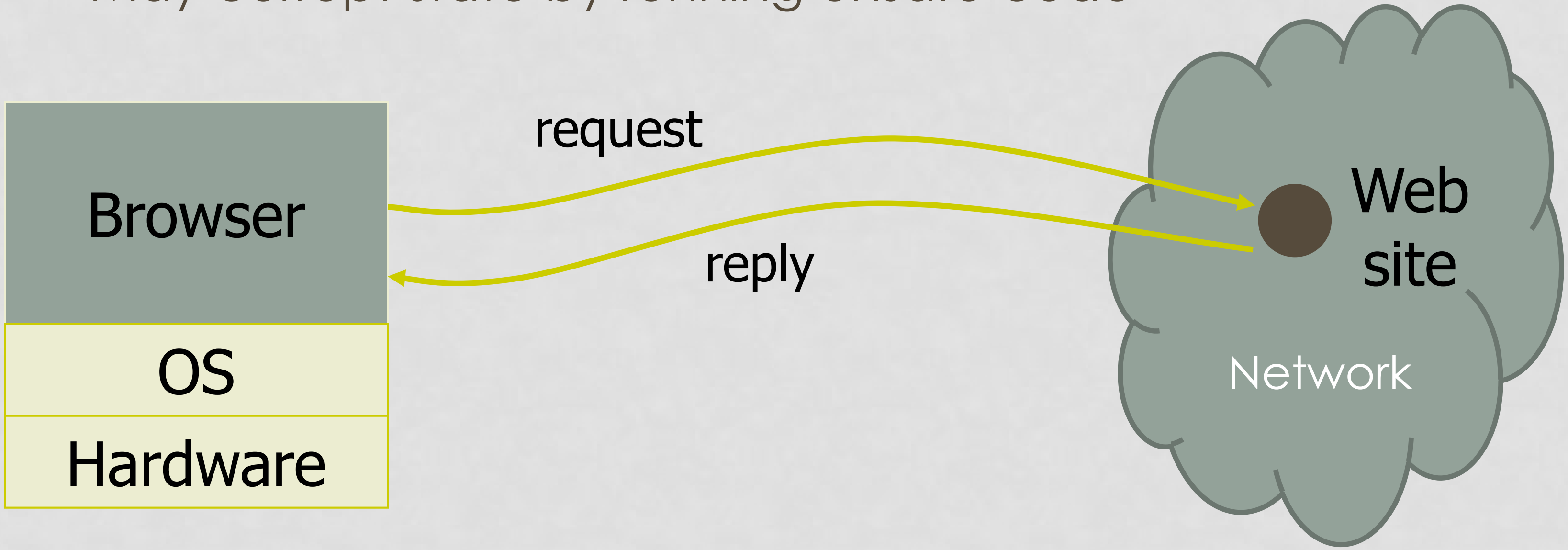
Browser Security

Why Browsers?

- Many attacks today exploits browser vulnerabilities.
- Browsers do not subject to perimeter protection.
- Browsers are complex
  - have many extensions
  - run downloaded code
- Important transactions are conducted over the browsers.

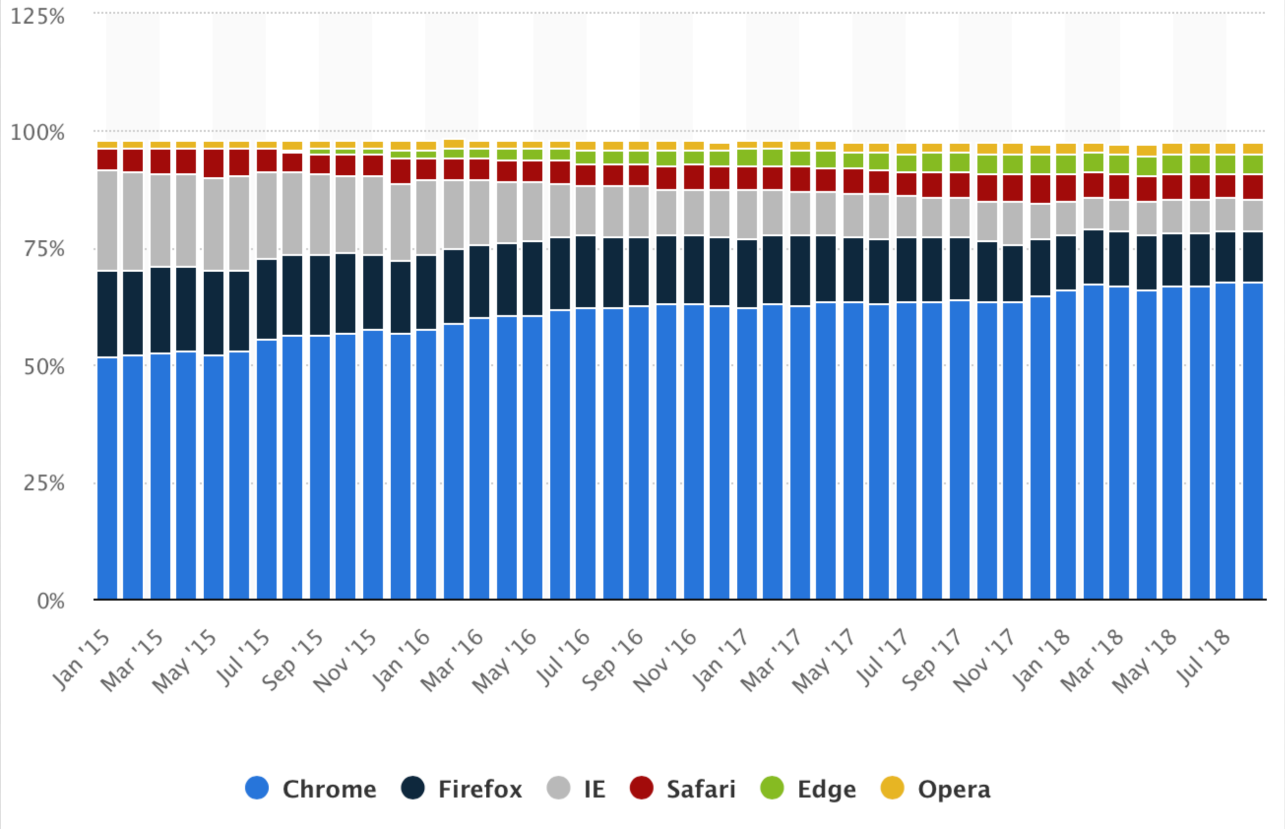
Browsers and Networks

- Browser sends requests
  - May reveal private information (in forms, cookies)
- Browser receives information, code
  - May corrupt state by running unsafe code

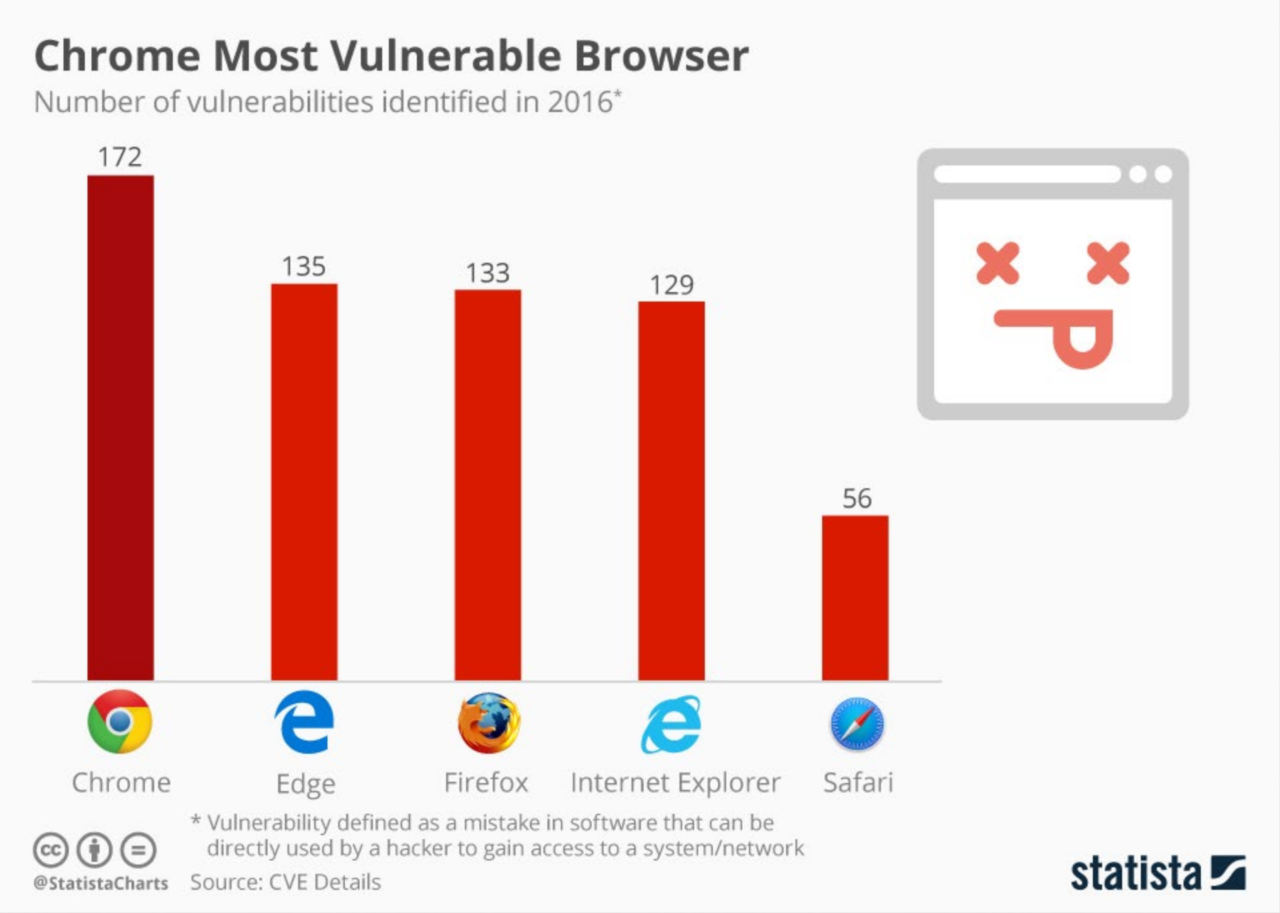


Most Recent Browsers Statistics

- Source: Statista



Browsers' Vulnerabilities



Browsers' Active Content

- Plug-ins:
  - Adobe Acrobat, Flash, Apple QuickTime, etc.
- Extensions.
- Active Code:
  - ActiveX
  - JavaScript

Security of Mobile Code

1. Sandboxing
   - Code executed in browser has only restricted access to OS, network
2. Isolation: the same-origin principle
   - Only the site that stores some information in the browser may later read or modify that information (or depend on it in any way).
3. Establish trust in the code
   - code digitally signed

SandBoxing

- Examine code before executing
  - Performs critical tests

- Interpret code and trap risky operations
  - Run-time tests
  - Security manager applies local access policy

- Security manager policy based on
  - Site that supplied the code
  - Code signing – who signed it?

Web Security

Understanding Web Applications

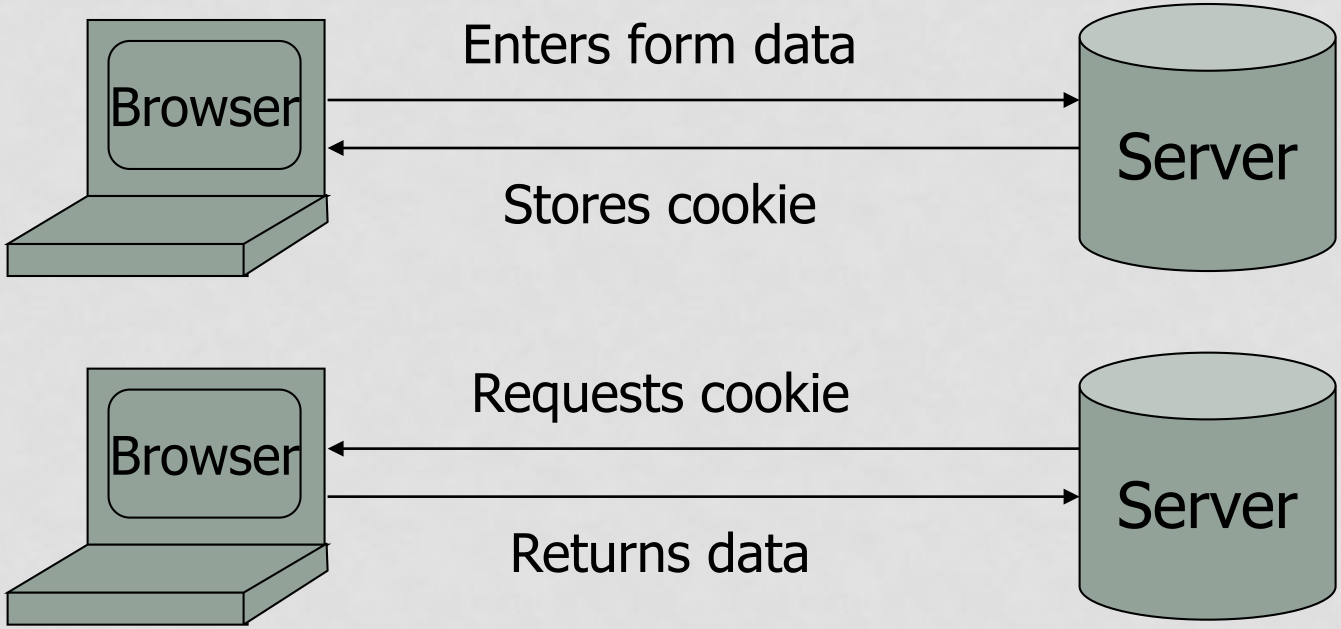
HTTP

- HTTP is a **stateless** protocol.
- Hosts do not need to retain information about users between requests
- Web applications must use alternative methods to track the user's progress from page to page
  - sending and receiving cookies
  - server side sessions, hidden variables and URL encoded parameters (such as /index.php?session_id=some_unique_session_code).

Cookies

- File created by a browser to store information on the client's computer.
- Can be read and written entirely on client side using Javascript.
- Used for authenticating, tracking, and maintaining specific information about user
- Security aspects
  - Data may be sensitive (security!)
  - May be used to gather information about specific users (privacy!).

State Maintenance with Cookies



Browsers and Cookies

- Cookie Same-origin ownership.
  - Once a cookie is saved on your computer, only the Web site that created the cookie can read it.

- Variations
  - Temporary cookies
    - Stored until you quit your browser
  - Persistent cookies
    - Remain until deleted or expire
  - Third-party cookies
    - Originates on or sent to a web site other than the one that provided the current page

3^{rd} Party Cookies – An Example

- Get a page from merchant.com
  - Contains <img src=http://doubleclick.com/advt.gif>
  - Image fetched from DoubleClick.com
    - DoubleClick knows IP address and page you were looking at.
- DoubleClick sends back a suitable advertisement
  - Stores a cookie that identifies "you" at DoubleClick
- Next time you get page with a doubleclick.com image
  - Your DoubleClick cookie is sent back to DoubleClick
  - DoubleClick could maintain the set of sites you viewed
  - Send back targeted advertising (and a new cookie)
- Cooperating sites
  - Can pass information to DoubleClick in URL, etc.

Cookies and Privacy

- Cookies maintain record of your browsing habits
  - Cookie stores information as set of name/value pairs
  - May include *any* information a web site knows about you
  - Sites track your activity from multiple visits to site
- Sites can share this information (e.g., DoubleClick).

Steps Forward

- DoNotTrack (donottrack.us):
  - Do Not Track is a technology and policy proposal that enables users to opt out of tracking by websites they do not visit, including:
    - analytics services,
    - advertising networks,
    - and social platforms.
- Cookie ClearingHouse (cch.law.stanford.edu):
  - Enforcement of DNT.

Web Security

Same Origin Policy and Scripting

Client Side Scripting – Revisit

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

- Script are powerful:
  - host access
    - read / write local files
  - webpage resources
    - cookies
    - Domain Object Model (DOM) objects.

HTML and Scripting

```
<html>
    …
   <P>
<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>
```

Same Origin Policy (SoP)

- The basic security model enforced in the browser
- Web users visits multiple websites simultaneously
- SoP isolates the scripts and resources downloaded from different origin
  - bank.com vs. evil.org
- Origin = domain name + protocol + port
  - All three must be equal for origin to be considered the same

Challenges to SoP

- 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

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
  - 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).

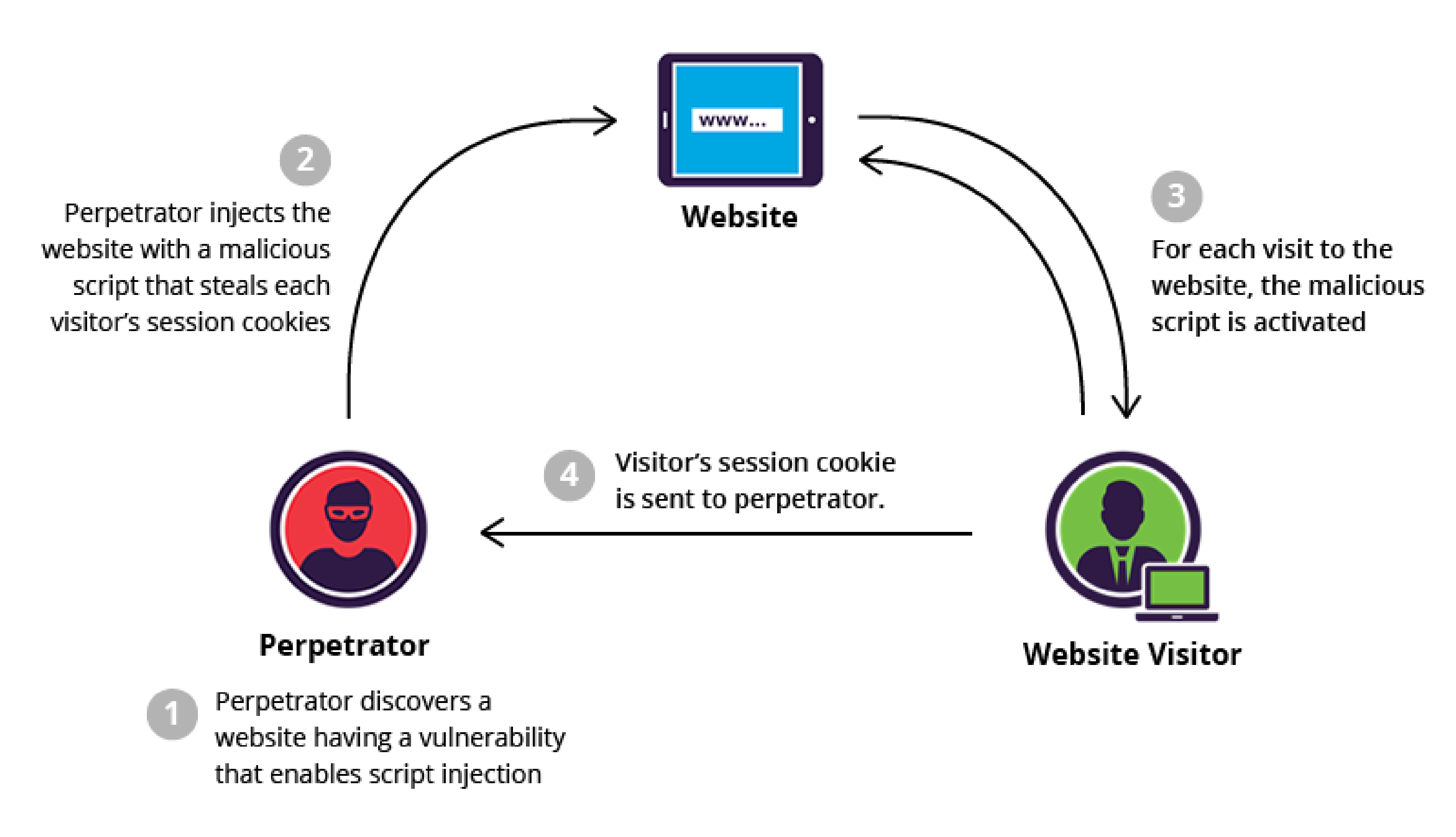
How Does XSS Works?

1. Everyone can post comments, which will be displayed to everyone who view the post

2. Attacker posts a malicious comment that includes scripts (which reads local authentication credentials and send of to the attacker)

3. Anyone who view the post can have local authentication cookies stolen

XSS Example



1 Perpetrator discovers a website having a vulnerability that enables script injection

2 Perpetrator injects the website with a malicious script that steals each visitor's session cookies

3 For each visit to the website, the malicious script is activated

4 Visitor's session cookie is sent to perpetrator.

Samy Worm – XSS Case Study

- In MySpace.com users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`
  - However, attacker find out that a way to include Javascript within CSS tags:

`<div style=“background:url(‘javascript:alert(1)’)”>`

And can hide **`“javascript”`** as **`“java\nscript”`**

- With careful javascript hacking:
  - Samy’s worm: infects anyone who visits an infected MySpace page – and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

XSS Prevention

- Input validation
  - Escaping and filtering
  - Eliminating script

Avoiding XSS (PHP)

- Main problem:
  - Input checking is difficult – many ways to inject scripts into HTML.

- Preprocess input from user before echoing it

- PHP: **htmlspecialchars**(string)

  & → &amp; " → &quot; ' → &#039;

  < → &lt; > → &gt;

  - **htmlspecialchars**(
    "<a href='test'>Test</a>", ENT_QUOTES);

    Outputs:

    &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;

Cross Site Request Forgery (CSRF or XSRF)

- Also known as **one click attack** or **session riding.**

- Transmits unauthorized commands from a user who has logged in to a website from another website.

CSRF Explained

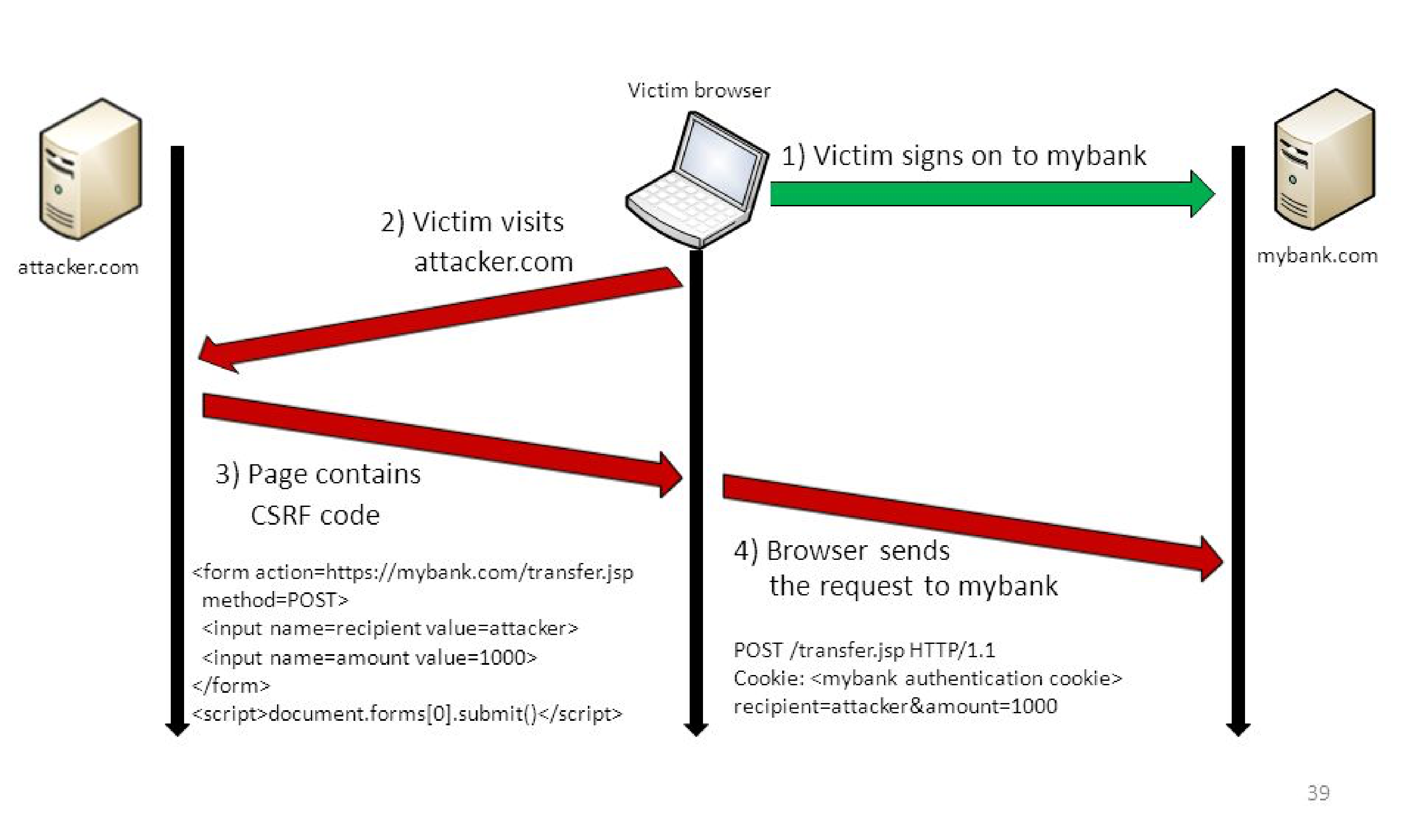
- Example:
  - User logs in to **bank.com**.
  - Session cookie remains in browser state
  - Then user visits another site containing:

```
<form  name=F  action=http://bank.com/BillPay.php>
<input  name=recipient   value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled

- Problem:
  - browser is a **confused deputy.**

CSRF



Victim browser

attacker.com

mybank.com

1) Victim signs on to mybank

2) Victim visits attacker.com

3) Page contains CSRF code

```
<form action=https://mybank.com/transfer.jsp
 method=POST>
 <input name=recipient value=attacker>
 <input name=amount value=1000>
</form>
<script>document.forms[0].submit()</script>
```

4) Browser sends the request to mybank

```
POST /transfer.jsp HTTP/1.1
Cookie: <mybank authentication cookie>
recipient=attacker&amount=1000
```

39

Preventing CSRF

- Server side protections:
  - Use cookie + hidden fields to authenticate
    - hidden fields values need to be unpredictable and user-specific
  - requires the body of the POST request to contain cookies

- User side protections:
  - Logging off one site before using others (usability!).

Web Security

SQL Injection

SQL Injection – A Typical Example

Phonebook Record Manager

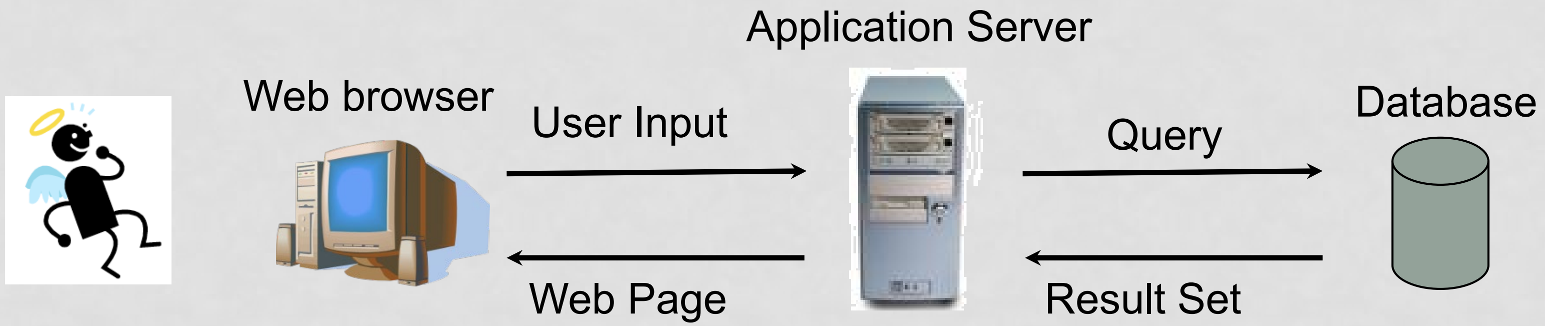
Username	John
Password	open_sesame

◉ **Display** ○ **Delete**

Submit

SELECT * FROM phonebook WHERE username =
'John' AND password =
'open_sesame'

John's phonebook entries are displayed



SQL Injection – A Typical Example 2

Phonebook Record Manager

Username John’ OR 1=1 --

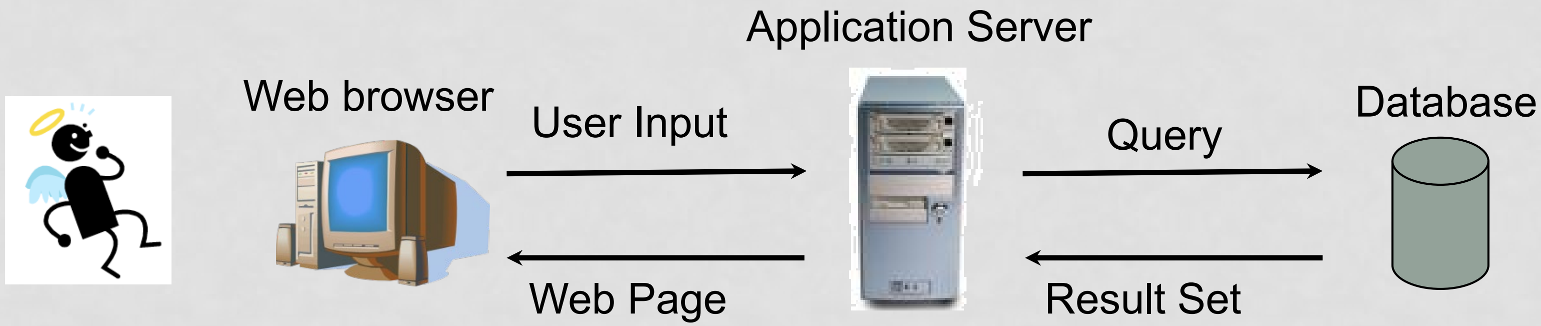
Password not needed

◉ **Display** ○ **Delete**

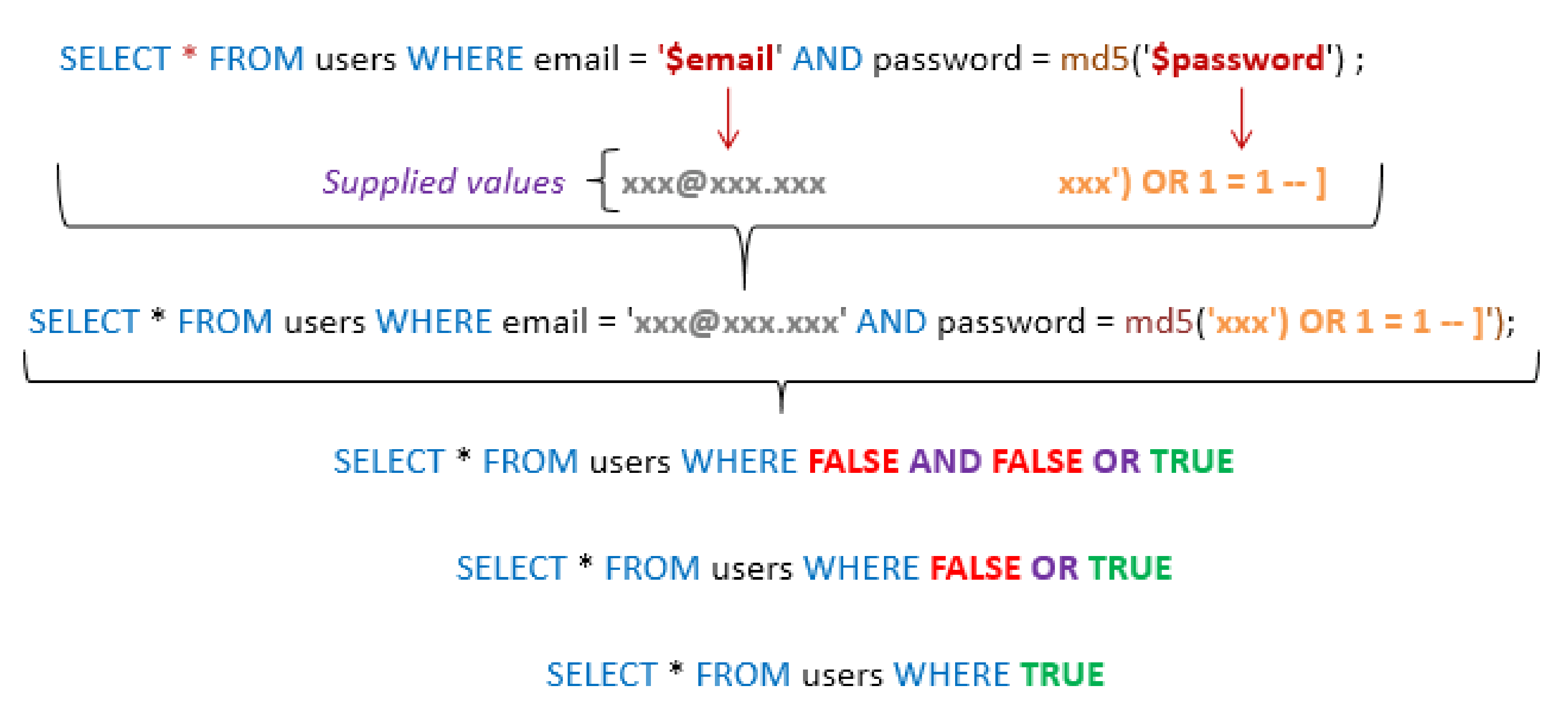
Submit

SELECT * FROM phonebook WHERE username = ‘John’ OR 1=1 --AND password = ‘not needed’

All phonebook entries are displayed



SQL Injection Example 3



```
SELECT * FROM users WHERE email = '$email' AND password = md5('$password') ;
```

Supplied values { xxx@xxx.xxx   xxx') OR 1 = 1 --]

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 -- ]');
```

```
SELECT * FROM users WHERE FALSE AND FALSE OR TRUE
```

```
SELECT * FROM users WHERE FALSE OR TRUE
```

```
SELECT * FROM users WHERE TRUE
```

Why SQL Injection Happens?

- SQL queries can be constructed by arbitrary sequences of programming constructs that involve string operations
  - Concatenation, substring ….
- Such construct also involve (untrusted) user inputs
  - Inputs should be mere “data”, but in case of SQL results in “code”.

SQL Injection Prevention

- Prepared Statements:
  - PREPARE stmt_name FROM " **SELECT * FROM phonebook WHERE username = ? AND password = ?”**

- Separates query structure from data

- Statements are NOT parsed for every user input

Next Lecture

- Security Standards and Principles

- Readings for next lecture:
  - Anderson’s Book – Sections 26.3
  - The Protection of Information in Computer Systems paper.