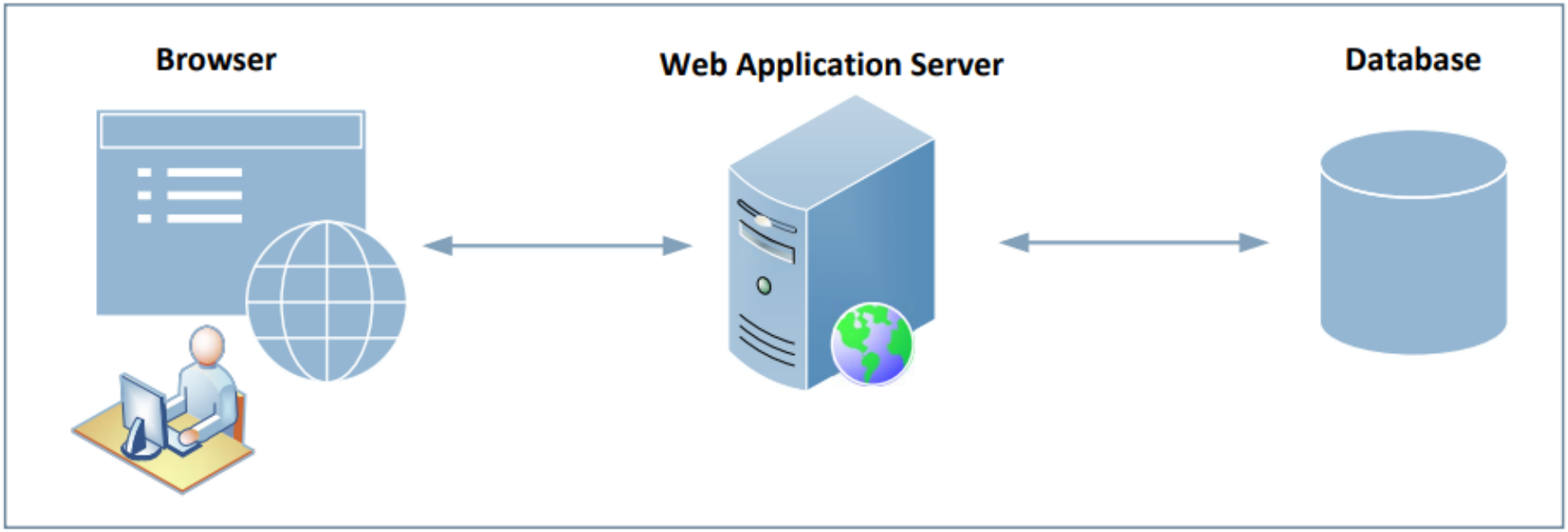
CS165 – Computer Security

Web Security

November 27, 2024

The Web Architecture



Web Application Security

- The **largest distributed system** in existence
- Multiple sources of threats, varying threat models
  - Clients (Many)
  - Servers (Can Be Multiple)
  - Web Applications
- Web Security: Securing the web infrastructure such that the integrity, confidentiality, and availability of content and user information is maintained

HTML

- Hypertext Markup Language
- For creating web pages
- Example

```
<html>
<body>
   <h1>Heading</h1>
   <p>This is a test.</p>
</body>
</html>
```

CSS: Cascading Style Sheets

- Specify the presentation style
- Separate content from the presentation style
- Example

```
<style type="text/css">
  .myclass   { background-color: yellow; }
  #myid { position:absolute; top:220px; left:700px; }
  body  { background-color: lightblue;
          margin-top:    50px; margin-bottom: 20px;
          margin-right:   0px; margin-left:   80px; }
  h1    { font-family: Arial, Helvetica, sans-serif; }
</style>
```

Dynamic Content

- Adobe Flash
- Microsoft Silverlight
- ActiveX
- Java applets
- **JavaScript**
- All run on the browser

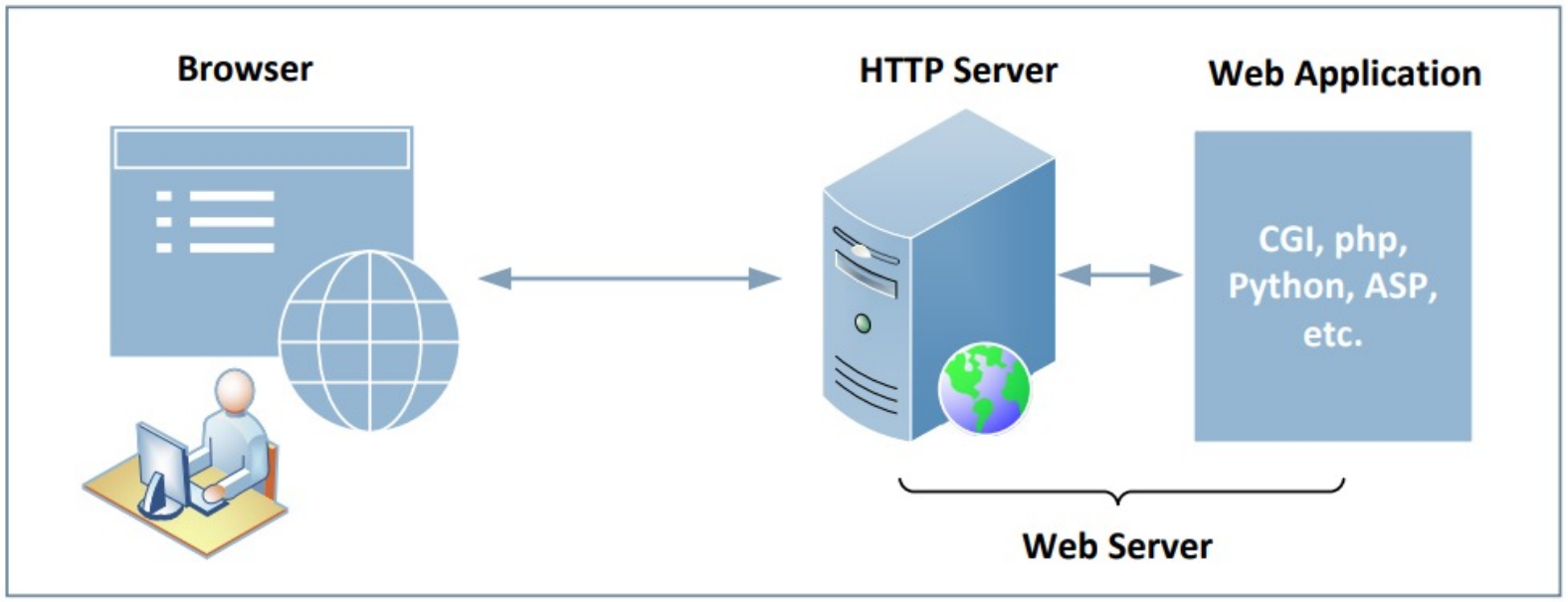
JavaScript

- Also known as ECMAScript
- Scripting language for web pages (run on browser)
- Different ways to include JavaScript code

```
<script>
  ... Code ...
</script>

<script src="myScript.js"></script>
<script src="https://www.example.com/myScript.js"></script>

<button type="button" onclick="myFunction()">Click it</button>
```

- Threat: Enables mixing of code and data

HTTP Server & Web Application Server



Cookies

- Web server is stateless
  - Does not maintain a long-term connection with the client
- HTTP Cookies: used to save information on the client side
  - Browsers save cookies
  - And attach cookies in every request

Session Cookies

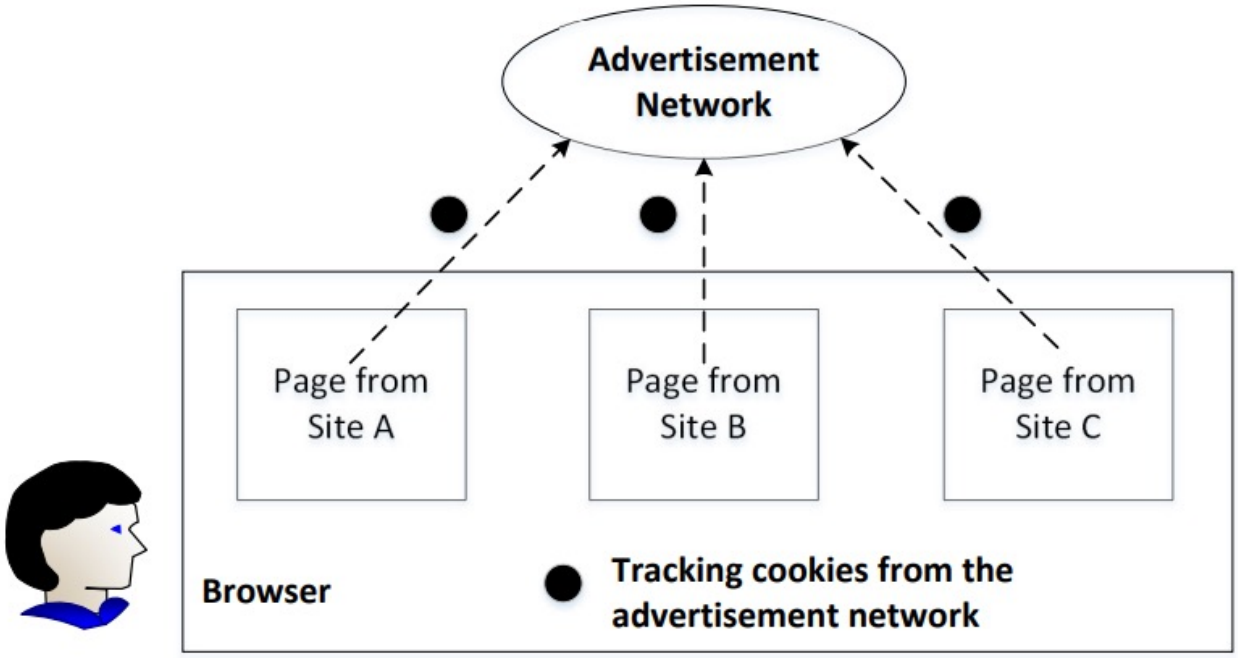
- A cookie: store session ID
- The session ID identifies a session
- Session data are typically maintained on the server
- Session is typically created after user login
  - Have the session ID = have the access
  - Security sensitive
  - ID: Random number

- What if the session data are stored on the client?
  - Need to ensure integrity via cryptographic signature

Cookie Management

- Browser stores cookies from multiple websites
  - Question: What is the threat model?

Tracking Using Cookies




```
<img src="advertisement network's website" width="1" height="1"/>
```

Prevent Tracking

- Using anonymous mode in browsing
- Block third-party cookies
  - First-party cookies are essential for browsing
  - Third-part cookies are mainly used for advertisement, information collection, etc.

Cookie Management

- Browser stores cookies from multiple websites
  - Question: What is the threat model?
- One website may steal cookies created by another
  - And other content

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins
- http://site.com vs https://site.com?

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins
- http://site.com vs https://site.com?
  - Different protocols are different origins

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins
- http://site.com vs https://site.com?
  - Different protocols are different origins
- http://site.com:80 vs http://site.com:8080?

Same Origin Policy

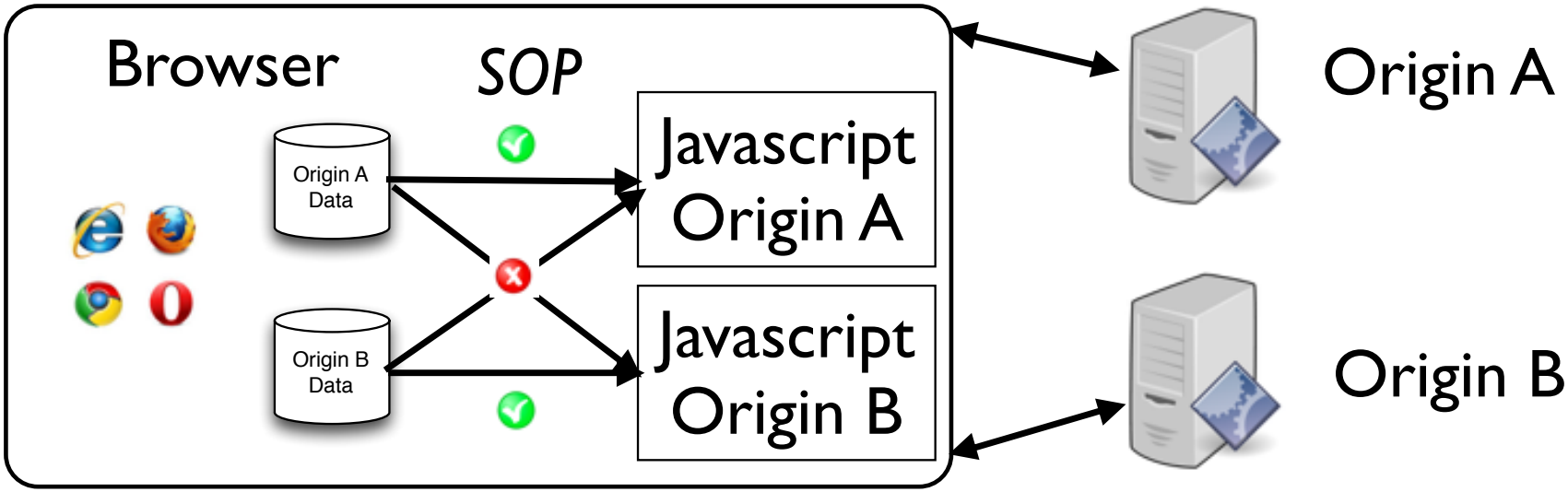
- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins
- http://site.com vs https://site.com?
  - Different protocols are different origins
- http://site.com:80 vs http://site.com:8080?
  - Different ports are different origins (applications)

Same Origin Policy

- A set of policies for isolating content across different sites (origins)
- What is an origin?
- site1.com vs site2.com?
  - Different hosts are different origins
- http://site.com vs https://site.com?
  - Different protocols are different origins
- http://site.com:80 vs http://site.com:8080?
  - Different ports are different origins (applications)
- Origin: host:protocol:port

Same Origin Policy

- *Principle:* Any active code from an origin can read only information stored in the browser that is from the same origin
  - Active code: Javascript, VBScript
  - Information: cookies, HTML responses, ...

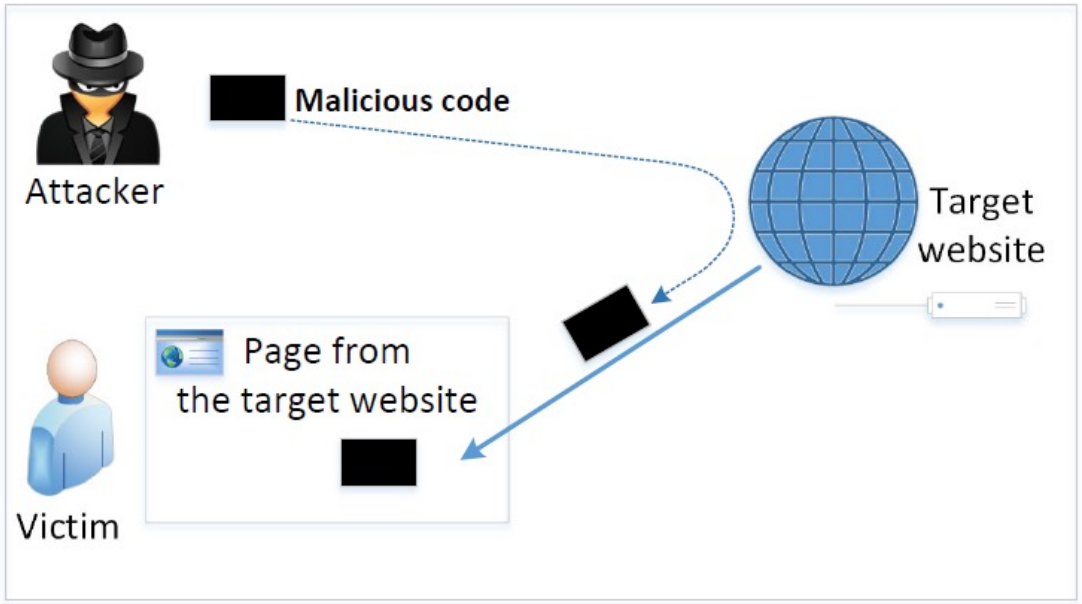


Relaxing the Restriction

- The same-origin policy is too restrictive
- CORS (Cross-Origin Resource Sharing)
  - Whitelist provided by server: grant permissions
- CORS policy on www.bank99.com

```php
<?php
 header("Access-Control-Allow-Origin: http://www.bank32.com");

 echo "Time from Bank99: ".date("h:i:sa")
?>
```

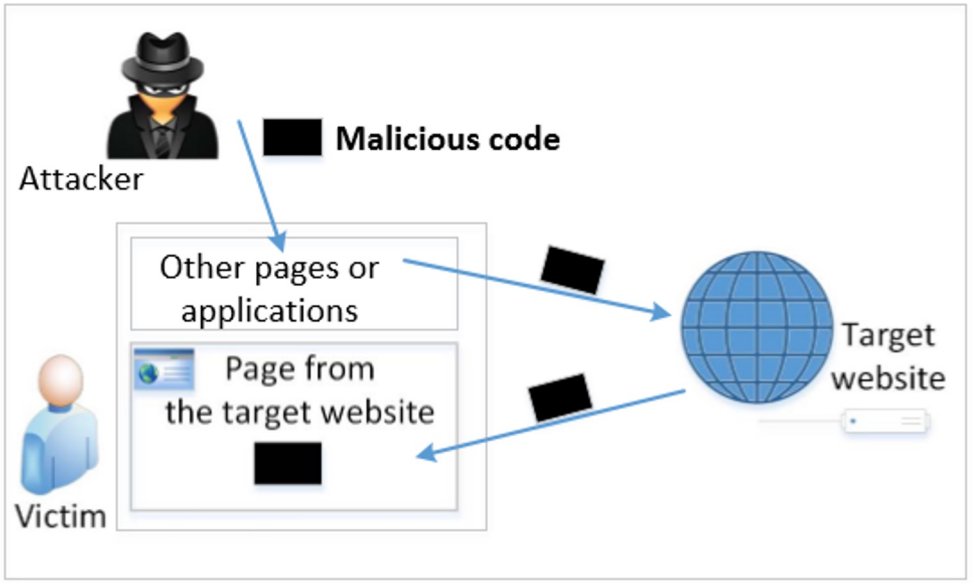
The Cross-Site Scripting Attack (XSS)



- Basically, code can do whatever the user can do inside the session.

- In XSS, an attacker injects his/her malicious code to the victim's browser via the target website.
- When code comes from a website, it is considered as trusted with respect to the website, so it can access and change the content on the pages, read cookies belonging to the website and sending out requests on behalf of the user.

Non-persistent (Reflected) XSS Attack



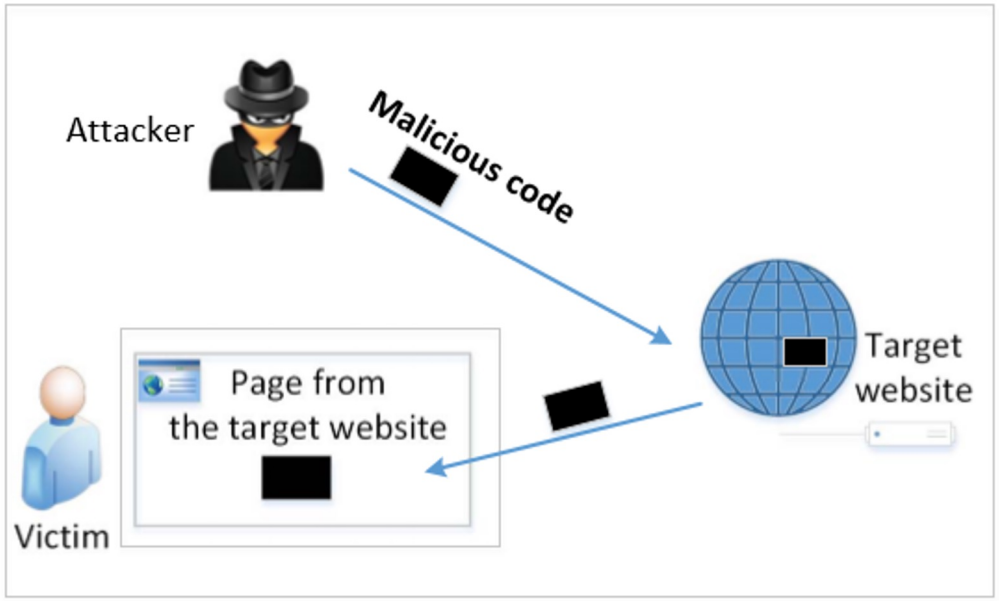
If a website with a reflective behavior takes user inputs, then :

- Attackers can put JavaScript code in the input, so when the input is reflected back, the JavaScript code will be injected into the web page from the website.

Non-persistent (Reflected) XSS Attack

- Assume a vulnerable service on website : http://www.example.com/search?input=word, where word is provided by the users.
- Now the attacker sends the following URL to the victim and tricks him to click the link: http://www.example.com/search?input=<script>alert("attack");</script>
- Once the victim clicks on this link, an HTTP GET request will be sent to the www.example.com web server, which returns a web page containing the search result, including the original input (word).
  - The input here is a JavaScript code which runs when the web page is loaded, producing a pop-up message on the victim's browser.

Persistent (Stored) XSS Attack



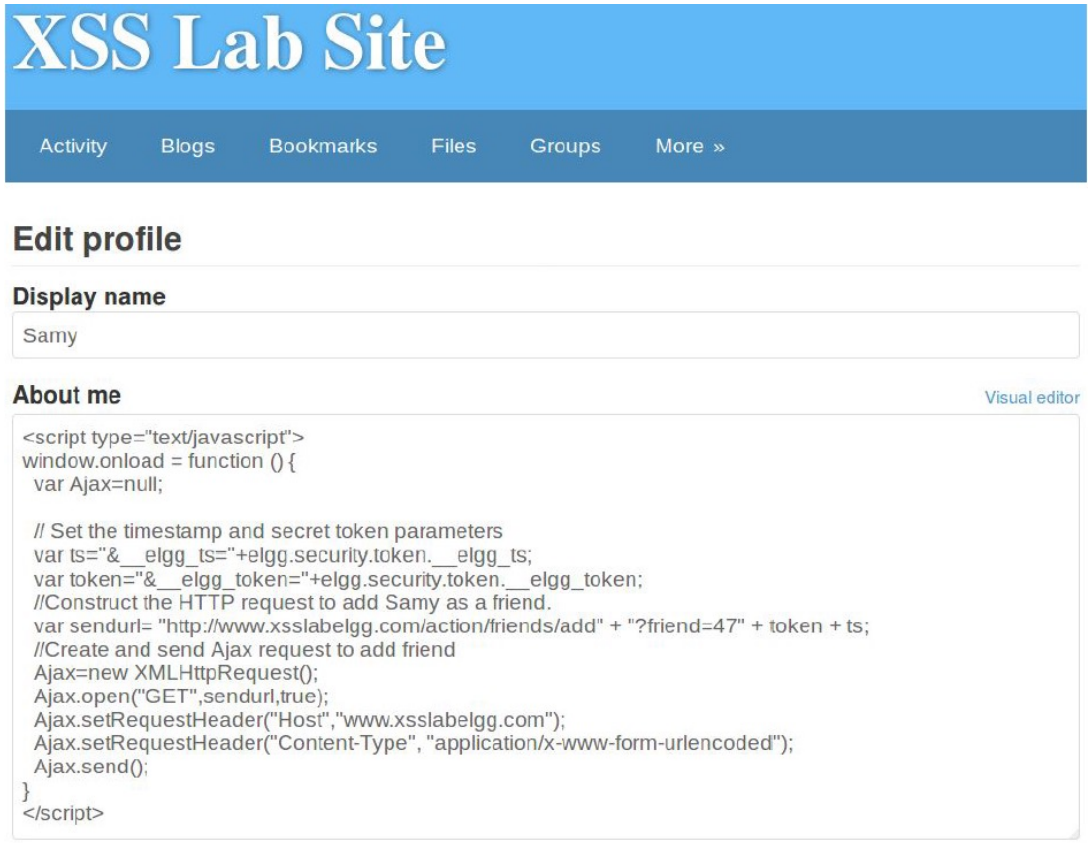
- Attackers directly send their data to a target website/server which stores the data in a persistent storage.
- If the website later sends the stored data to other users, it creates a channel between the users and the attackers.

Example : User profile in a social network is a channel as it is set by one user and viewed by another.

Persistent and Reflected XSS Attack

- These channels are supposed to be data channels.
- But data provided by users can contain HTML markups and JavaScript code.
- If the input is not sanitized properly by the website, it is sent to other users' browsers through the channel and gets executed by the browsers.
- Browsers consider it like any other code coming from the website. Therefore, the code is given the same privileges as that from the website.

Add Samy As a Friend (via XSS)



- Samy puts the script in the "About Me" section of his profile.
- After that, let's login as "Alice" and visit Samy's profile.
- JavaScript code will be run and not displayed to Alice.
- The code sends an add-friend request to the server.
- If we check Alice's friends list, Samy is added.

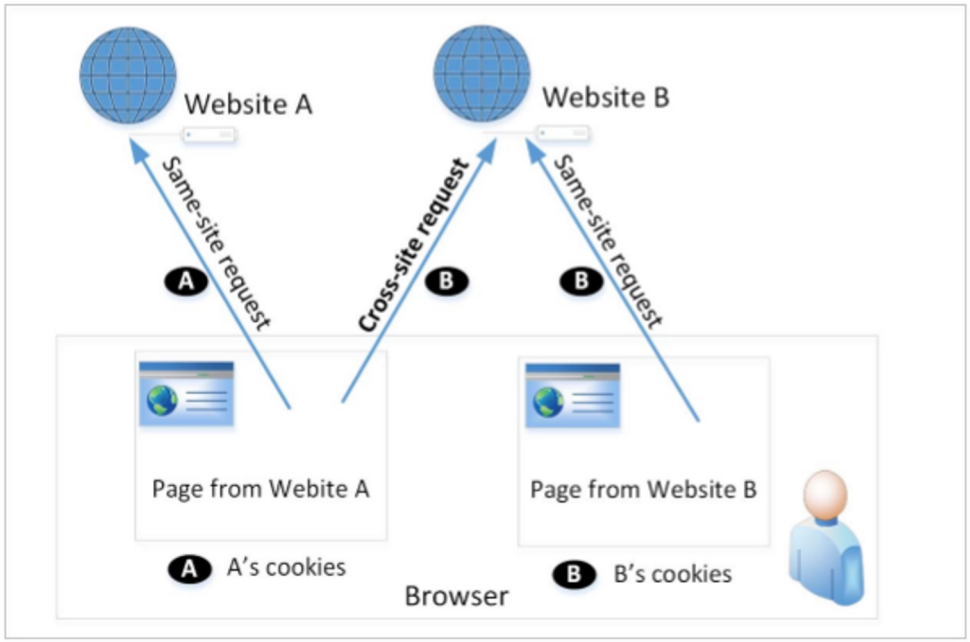
Damage Caused by XSS

- Web defacing: JavaScript code can access the data stored inside the hosting page (DOM).
  - Therefore, the injected JavaScript code can make arbitrary changes to the page. Example: JavaScript code can change a news article page to something fake or change some pictures on the page.
- Spoofing requests: The injected JavaScript code can send HTTP requests to the server on behalf of the user.
- Stealing information: The injected JavaScript code can also steal victim's private data including the session cookies, personal data displayed on the web page, data stored locally by the web application.

Countermeasures for XSS

- Sanitize inputs: Do not allow insertion of JavaScript (code) in arguments and web pages (data)
  - **PHP module HTMLawed** (on server)
  - Highly customizable PHP script to sanitize HTML against XSS attacks.
- Do not execute: Change web pages to prevent the execution of code where data is expected
  - **PHP function htmlspecialchars** (on server)
  - Encode data provided by users, s.t., JavaScript code in user's inputs will be interpreted by browsers only as strings and not as code.

Cross-Site Requests



- When a page from a website sends an HTTP request back to the website, it is called same-site request.
- If a request is sent to a different website, it is called cross-site request because where the page comes from and where the request goes are different.

E.g., a webpage (not Facebook) can include a Facebook link, so when users click on the link, HTTP request is sent to Facebook.

Cross-Site Request Forgery

- When a request is sent to example.com from a page coming from example.com, the browser attaches all the cookies belonging to example.com.
- Now, when a request is sent to example.com from another site (different from example.com), the browser will attach the cookies too.
- Because of above behavior of the browsers, the server cannot distinguish between the same-site and cross-site requests
- It is possible for third-party websites to forge requests that are exactly the same as the same-site requests.
- This is called Cross-Site Request Forgery (CSRF).

Cross-Site Request Forgery: Approach

- **Environment Setup:**
  - Target website
  - Victim user who has an active session on the target website
  - Malicious website attacker

- **Steps:**
  - The attacker crafts a webpage that can forge a cross-site request to be sent to the targeted website.
  - The attacker needs to attract the victim user to visit its malicious website.
  - The attacker performs requests on the target website as the victim.

How to Prevent CSRF

- The server cannot distinguish whether a request is cross-site or same-site
  - Same-site request: coming from the server's own page. Trusted.
  - Cross-site request: coming from other site's pages. Not Trusted.
- We cannot treat these two types of requests the same

How to Prevent CSRF

- The server cannot distinguish whether a request is cross-site or same-site
  - Same-site request: coming from the server's own page. Trusted.
  - Cross-site request: coming from other site's pages. Not Trusted.
- We cannot treat these two types of requests the same
- Does the browser know the difference?

How to Prevent CSRF

- The server cannot distinguish whether a request is cross-site or same-site
  - Same-site request: coming from the server's own page. Trusted.
  - Cross-site request: coming from other site's pages. Not Trusted.
- We cannot treat these two types of requests the same
- Does the browser know the difference?
  - Of course. The browser knows from which page a request is generated.

How to Prevent CSRF

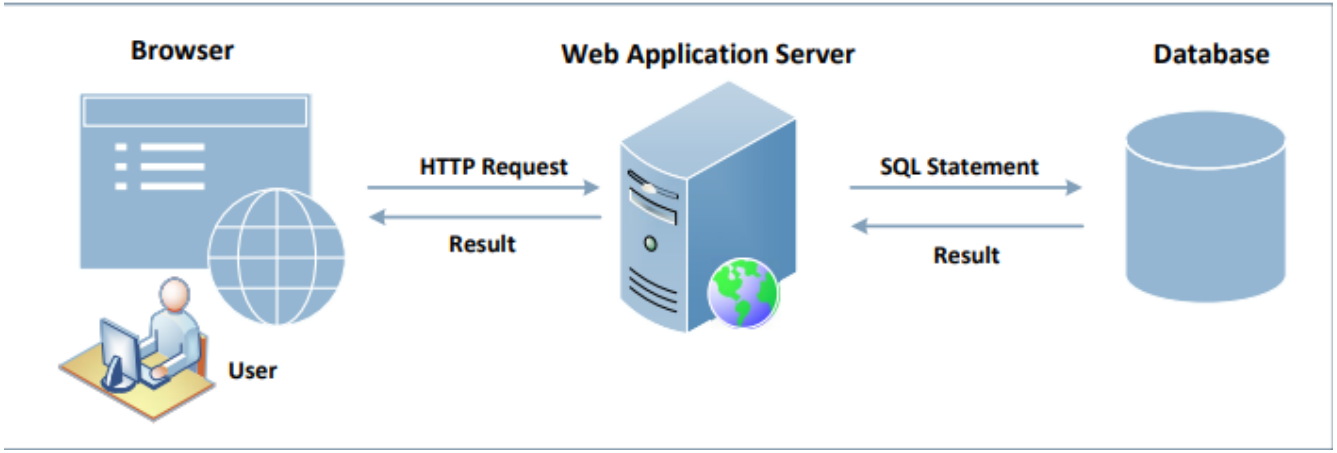
- The server cannot distinguish whether a request is cross-site or same-site
  - Same-site request: coming from the server's own page. Trusted.
  - Cross-site request: coming from other site's pages. Not Trusted.
- We cannot treat these two types of requests the same
- Does the browser know the difference?
  - Of course. The browser knows from which page a request is generated.
- Countermeasures
  - Referrer header (browser's help)
  - Same-site cookie (browser's help)
  - Secret token (the server helps itself to defend against CSRF)

Countermeasures: Same-Site Cookies

- A special type of cookie in browsers like Chrome and Opera, which provide a special attribute to cookies called SameSite.
- This attribute is set by the servers and it tells the browsers whether a cookie should be attached to a cross-site request or not.
- Cookies with this attribute are always sent along with same-site requests, but whether they are sent along with cross-site depends on the value of this attribute.
- Values
  - Strict (Not sent along with cross-site requests)
  - Lax (Sent with cross-site requests)

Exploiting Database of a Web Application

- A typical web application consists of three major components



- SQL Injection attacks can cause damage to the database.
  - As we notice in the figure, the users do not directly interact with the database but through a web server. If this channel is not implemented properly, malicious users can attack the database.

Launching SQL Injection Attacks

- Everything provided by user will become part of the SQL statement. Is it possible for a user to change the meaning of the SQL statement?
- The intention of the web app developer by the following is for the user to provide some data for the blank areas.

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid=' [                ] ' and password=' [                ] '
```

- Assume that a user inputs a random string in the password entry and types “EID5002’#” in the eid entry. The SQL statement will become the following

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'EID5002' #' and password='xyz'
```

Launching SQL Injection Attacks

- Everything from the # sign to the end of line is considered a comment. The SQL statement will be equivalent to the following:

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'EID5002'
```

- The above statement will return the name, salary and SSN of the employee whose EID is EID5002 even though the user doesn't know the employee's password. This is security breach.
- Let's see if a user can get all the records from the database, assuming that we don't know all the EID's in the database.
- We need to create a predicate for WHERE clause so that it is true for all records.

```
SELECT Name, Salary, SSN
FROM employee
WHERE eid= 'a' OR 1=1
```

SQL Injection Defenses

- Fundament cause of SQL injection: mixing data and code
- Fundament solution: separate data and code.
- Main Idea: Sending code and data in separate channels to the database server. This way the database server knows not to retrieve any code from the data channel.
  - How: using prepared statement
- Prepared Statement: Using prepared statements, we send an SQL statement template to the database, with certain values (the data) called parameters left unspecified.
  - The database performs query optimization on the SQL statement template and stores the result without executing it.
  - We later bind the data (from browser) to the prepared statement

Countermeasures: Prepared Statement

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= '$eid' and password='$pwd'";
$result = $conn->query($sql);
```

- The vulnerable version: code and data are mixed together.

Using prepared statements, we separate code and data.

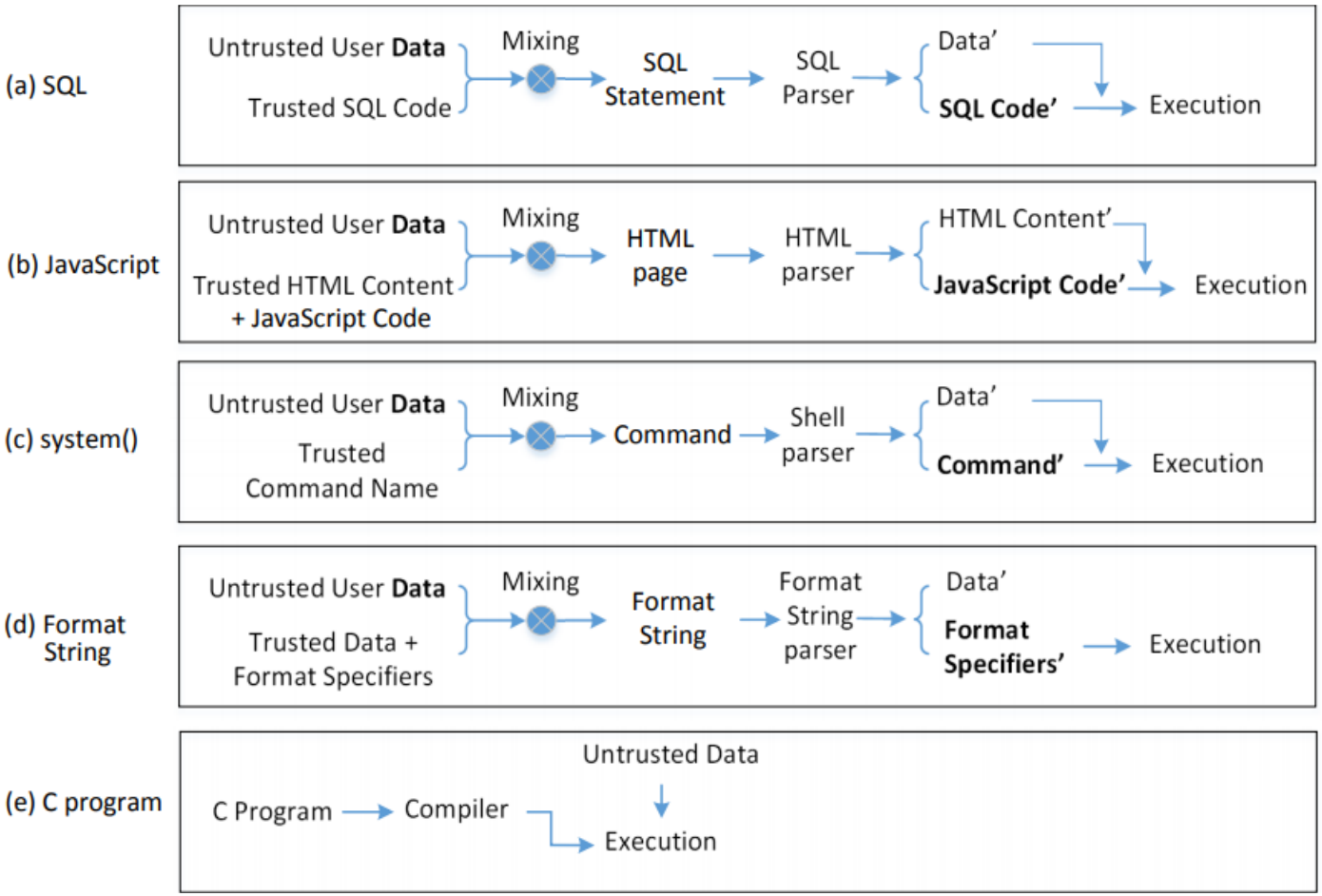
```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";                    ①

if ($stmt = $conn->prepare($sql)) {                      ②
   $stmt->bind_param("ss", $eid, $pwd);                  ③
   $stmt->execute();                                     ④

   $stmt->bind_result($name, $salary, $ssn);             ⑤
   while ($stmt->fetch()) {                              ⑥
      printf ("%s %s %s\n", $name, $salary, $ssn);
   }
}
```

Send code

Send data

Start execution

A Fundamental Cause



Mixing data and code together is the cause of several types of vulnerabilities and attacks including SQL Injection attack, XSS attack, attacks on the return address and the heap attacks in P3.

Conclusions

- The “web” is the most complex distributed system in the world
  - Manifest in a variety of web applications
- Web protocols (http/https) are inherently stateless, so web applications store state
  - On server and client (cookies, JavaScript objects, etc.)
- XSS and XSRF attacks
  - Confuse clients and servers about sources of requests
- +SQL Injection, mix of data and code is problem

Questions