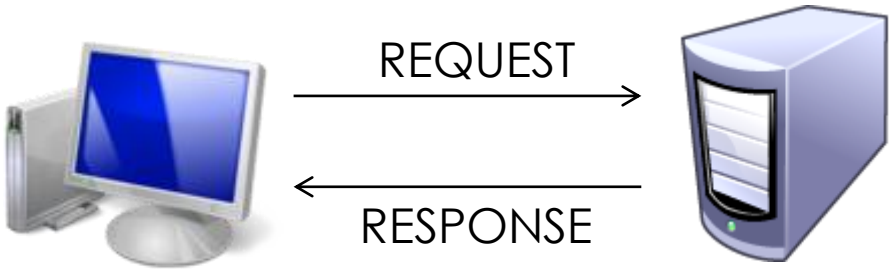
Foundations of Cybersecurity
Web Security Basics

Michele La Manna
Dept. of Information Engineering
University of Pisa
michele.lamanna@phd.unipi.it
Version: 2022-05-18
(Original slides thanks to Eng. Pericle Perazzo, PhD)

HTTP

UNIVERSITÀ DI PISA

- Request/response messages
- **Stateless**
  - State delegated to web application
  - Server-side state (session variables, database)
  - Client-side state (cookies)



Web Security

UNIVERSITÀ DI PISA

- Users can submit arbitrary input
  - Many inputs (requested URL, params, headers, body)
  - Requests can arrive in any sequence
- Attackers do not use (only) browsers
  - Clients can change anything in HTTP request
  - Clients can read anything in HTTP response (cookies, hidden fields, headers)
  - Requests can arrive with quantity/rate impossible with browser

Catch the design flaws!

UNIVERSITÀ DI PISA

- Online shopping website
- Order placing procedure:
  1. User browses catalog, adds items to shopping basket by clicking «Add». Shopping basket is stored on a cookie to save server storage.
  2. If user adds a given combination of items, a special «discount item» is added
  3. User clicks on «Finalize order», which stores basket on server and leads to a page asking credit card info
  4. User clicks on «Buy», which stores credit card info and leads to a page asking shipping info
  5. User clicks on «Ship order», which stores shipping info and leads back to product catalog

Never Trust Cookies

UNIVERSITÀ DI PISA

- 1° flaw: User can add non-authorized discount items
- If the discount item has secret ID, user can obtain it by adding the correct item combination, then delete the items
- NEVER TRUST Client-side state!
- Security-critical info on server-side state.

Never Trust Request Sequence

- 2° flaw: User can skip sending the credit card info, by jumping directly to the URL used to send shipping info
- No trusted «control flow» between successive HTTP requests
- In multi-step procedures, every step must check the precedent steps

Web Security Basics

SQL INJECTION

Catch the bug!

UNIVERSITÀ DI PISA

```
$name = $_POST['username'];
$pwdHash = password_hash($_POST['pwd'], PASSWORD_DEFAULT);
$res = mysqli_query($link,
  "SELECT * FROM users WHERE name = '$name' AND pwdHash='$pwdHash'"
);
if (!$res) { die('Query error'); }
$row = mysqli_fetch_row($res);
if (!$row) { die('Invalid user ID or password'); }
// … begin session
```

$_POST['username'] = JohnDoe
$_POST['pwd'] = pa55w0rd

SELECT * FROM users WHERE name = 'JohnDoe' AND pwdHash='$2y$10$[…]'

SQL Injection

UNIVERSITÀ DI PISA

```
SELECT * FROM users WHERE name = '$name' AND pwdHash='$pwdHash'
```

$_POST['username'] = admin' --
$_POST['pwd'] = letmein

```
SELECT * FROM users WHERE name = 'admin' -- ' AND pwdHash='[...]'
```

ignored as a comment

Open Web Application Security Project (OWASP)

OWASP Top 10 2017		change	OWASP Top 10 2021 proposal	
A1	Injections	**as is**	A1	Injections
A2	Broken Authentication	**as is**	A2	Broken Authentication
A3	Sensitive Data Exposure	**down 1**	A3	Cross-Site Scripting (XSS)
A4	XML eXternal Entities (XXE)	**down 1 + A8**	A4	Sensitive Data Exposure
A5	Broken Access Control	**down 1**	A5	Insecure Deserialization **(merged with XXE)**
A6	Security Misconfiguration	**down 4**	A6	Broken Access Control
A7	Cross-Site Scripting (XSS)	**up 4**	A7	Insufficient Logging & Monitoring
A8	Insecure Deserialization	**up 3 + A4**	A8	**NEW:** Server Side Request Forgery (SSRF)
A9	Known Vulnerabilities	**as is**	A9	Known Vulnerabilities
A10	Insufficient Logging & Monitoring	**up 3**	A10	Security Misconfiguration

SQL Injection

UNIVERSITÀ DI PISA

- Risks: Bypassing authentication, escalating privileges, stealing data, adding or modifying data, partially or totally deleting a database.
- Interpreted languages (SQL, LDAP, XPath, etc.): mix of programmer instructions + user input
- In case of «bad» mix: part of user input interpreted as code
- Injected code executed as legitimate programmer code

Tautology

UNIVERSITÀ DI PISA

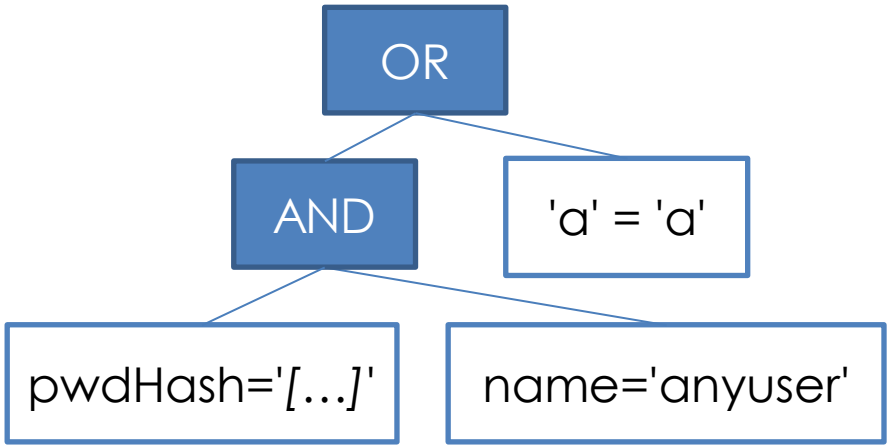
- Makes a WHERE clause always true

```
SELECT * FROM users WHERE pwdHash = '$pwdHash' AND name='$name'
```

$name = anyuser' OR 'a'='a

```
SELECT * FROM users WHERE pwdHash = '[…]' AND name='anyuser' OR 'a' = 'a'
```

quote balancing (alternative to line commenting)



same effect of:
$name = anyuser' OR 1=1 --

Countermeasures

UNIVERSITÀ DI PISA

- Rejecting inputs by whitelisting (good practice, but false positives)
- Input escaping

$name = anyuser' OR 'a'='a

SELECT * FROM users WHERE pwdHash = '*[…]*' AND name='anyuser\' OR \'a\' = \'a'

Input Escaping

UNIVERSITÀ DI PISA

```
$name = $_POST['username'];
$pwdHash = password_hash($_POST['pwd'], PASSWORD_DEFAULT);
$res = mysqli_query($link,
  "SELECT * FROM users
  WHERE name = 'mysql_real_escape_string($name,$link)'
    AND pwdHash='mysql_real_escape_string($pwdHash,$link)'"
);
if (!$res) { die('Query error'); }
$row = mysqli_fetch_row($res);
if (!$row) { die('Invalid user ID or password'); }
// … begin session
```

- Escape each time tainted data is concatenated (error prone!)

Prepared Statements

UNIVERSITÀ DI PISA

- Prepared statements (better solution)
  - Query is built in two stages (code, parameters)
  - Available since PHP 5.0, with «mysqli_*()» APIs

```
$name = $_POST['name'];
$pwdHash = password_hash($_POST['pwd'], PASSWORD_DEFAULT);
$stmt = mysqli_stmt_init($link);
mysqli_stmt_prepare($stmt, "SELECT * FROM users WHERE name=?
AND pwdHash=?");
mysqli_stmt_bind_param($stmt, 'ss', $name, $pwdHash);
if(!mysqli_stmt_execute($stmt)) { die('Query error'); }
$res = mysqli_stmt_get_result($stmt);
$row = mysqli_fetch_row($res);
if (!$row) { die('Invalid user ID or password'); }
mysqli_stmt_close($stmt);
// … begin session
```

Read the DOCs!

- https://www.php.net/manual/en/mysqli.stmt-init.php
- https://www.php.net/manual/en/mysqli-stmt.prepare.php
- https://www.php.net/manual/en/mysqli-stmt.bind-param.php
- https://www.php.net/manual/en/mysqli-stmt.execute.php
- https://www.php.net/manual/en/mysqli-stmt.get-result.php
- https://www.php.net/manual/en/mysqli-result.fetch-row.php
- https://www.php.net/manual/en/mysqli-stmt.close.php

Web Security Basics

CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting

UNIVERSITÀ DI PISA

- E.g., display a dynamic error message
  - Error message taken from GET param

```
echo "<p>ERROR: $_GET['msg']</p>";
```

[visit:]
http://mysite.com/show_error.php?msg=Sorry+Page+Not+Found

```
<p>ERROR: Sorry Page Not Found</p>
```

Cross-Site Scripting

UNIVERSITÀ DI PISA

```
echo "<p>ERROR: $_GET['msg']</p>";
```

[visit:]
http://mysite.com/show_error.php?msg=%3Cscript%3Ealert(1)%3C%2Fscript%3E

$_GET['msg'] = <script>alert(1)</script>

```
<p>ERROR: <script>alert(1)</script></p>
```

Executed by the client!

Cross-Site Scripting

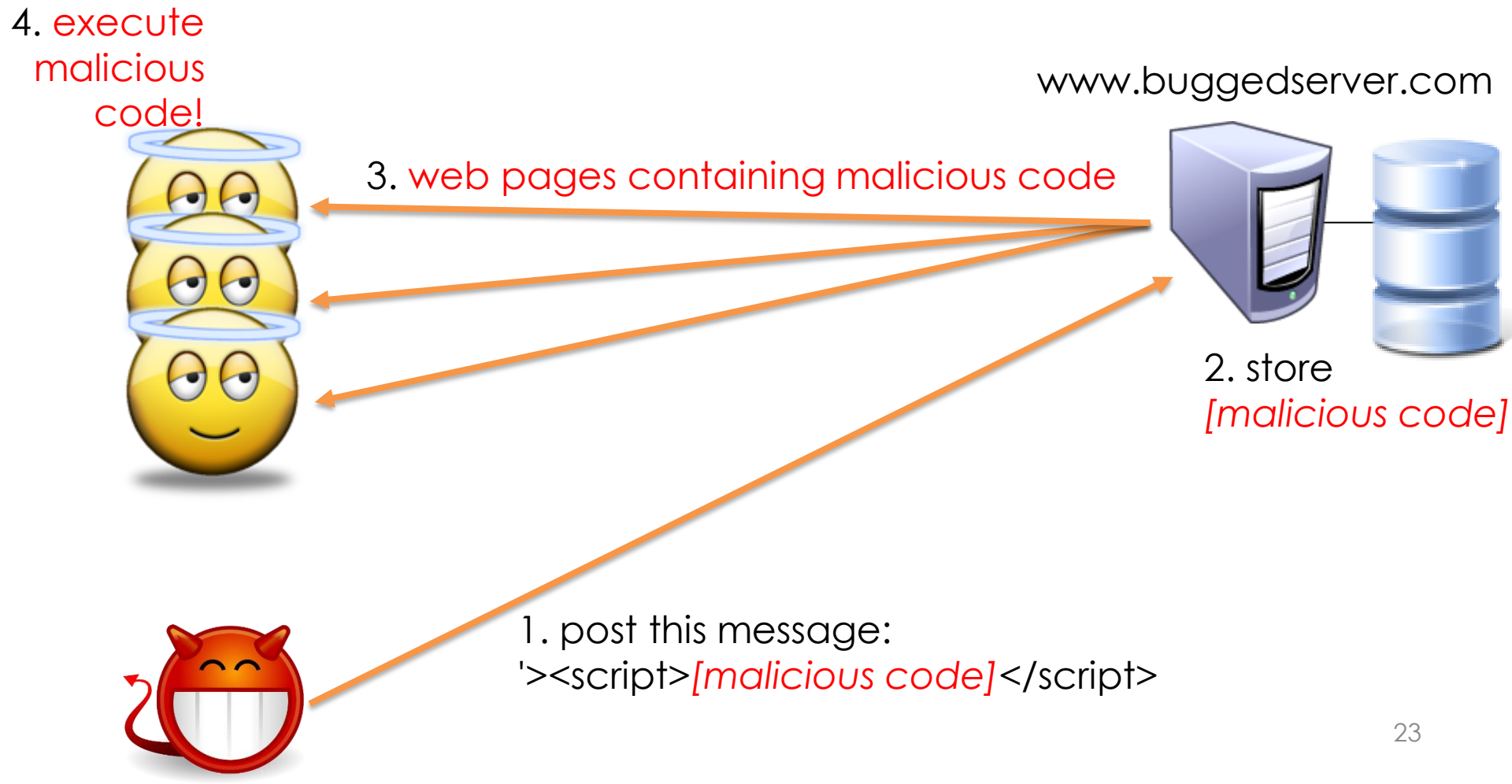
UNIVERSITÀ DI PISA

- Reflected XSS:

Cross-Site Scripting

UNIVERSITÀ DI PISA

- Stored XSS:



Countermeasure

UNIVERSITÀ DI PISA

```
$escp_msg = htmlspecialchars($_GET['msg'], ENT_QUOTES);
echo "<p>ERROR: $escp_msg</p>";
```

[visit:]
http://mysite.com/show_error.php?msg=%3Cscript%3Eal
ert(1)%3C%2Fscript%3E
$_GET['msg'] = <script>alert(1)</script>
$escp_msg = ⁢script>alert(1)⁢/script>

```
<p>ERROR: &it;script&gt;alert(1)&it;/script&gt; </p>
```

Read the DOCs:
https://www.php.net/manual/en/function.htmlspecialchars.php

Cool laboratory coming up!

You will now test your skills of SQL injections and XSS attacks in a cyber-range platform.

You can use this set of slides (available on MS TEAMS).

We have a limited amount of virtual resources so we have to arrange some working-groups right now.