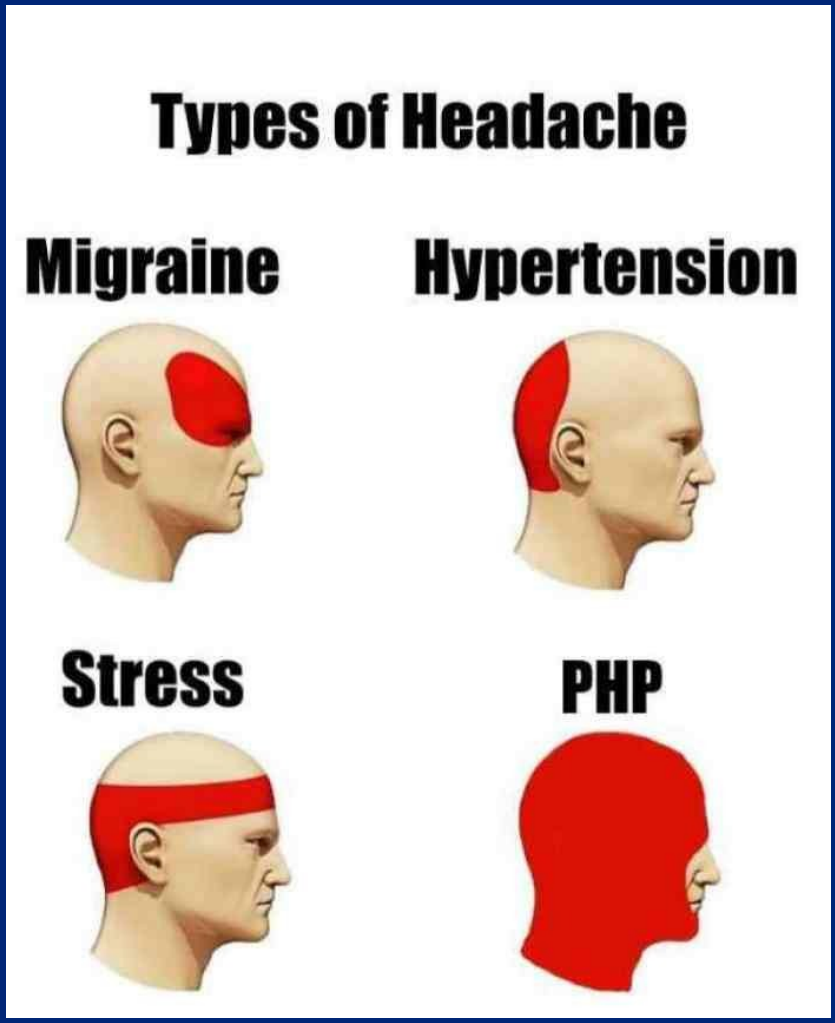
Università di Genova

PHP (2)



Marina Ribaudo, marina.ribaudo@unige.it

Gestione delle password

- Le password degli utenti non si devono **MAI** memorizzare in chiaro
  - Molti attacchi sul web hanno lo scopo di rubare le password degli utenti https://haveibeenpwned.com/
  - In caso di furto, bisogna garantire che l'attaccante non riesca a decifrare le password
  - Oggi - per rispettare le regole del GDPR - bisogna proteggere i dati sensibili (e le password lo sono) https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

PHP: gestione delle password


```
/* User's password. */
$password = 'mysecretpassword';
/* MD5 hash */
$hash = md5($password);
/* SHA1 hash */
$hash = sha1($password);
```

PHP: gestione delle password


```
/* User's password. */
$password = 'mysecretpassword';
/* MD5 hash */
$hash = md5($password);
/* SHA1 hash */
$hash = sha1($password);
```

Warning It is not recommended to use this function to secure passwords, due to the fast nature of this hashing algorithm. See the Password Hashing FAQ for details and best practices.

PHP: gestione delle password

- Gli algoritmi **MD5** e **SHA** sono **troppo deboli** per la potenza di calcolo odierna
- Gli hash prodotti da questi algorimti sono vulnerabili ad attacchi “rainbow table” e ad attacchi a dizionario
  - https://crackstation.net/ (a volte basta Google!)
  - https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

PHP: gestione delle password


```
/* User's password. */
$password = 'mysecretpassword';
$hash = password_hash($password, PASSWORD_DEFAULT);
```

- Il secondo parametro specifica l'algoritmo di hash da usare
- Questa funzione usa un "sale" che viene aggiunto alla password per evitare attacchi rainbow table o a dizionario
- Il "sale" è una **stringa casuale** generata da PHP
- Per verificare le password degli utenti si deve usare la funzione **password_verify**()

https://www.php.net/manual/en/function.password-hash.php

PHP: dati in arrivo dal client

“ … all input is evil until proven otherwise …”

I dati in arrivo dal client devono essere **validati**!

PHP ha varie funzioni che permettono di testare il contenuto dei dati forniti in input

Il test dipende dalle abilità del programmatore...

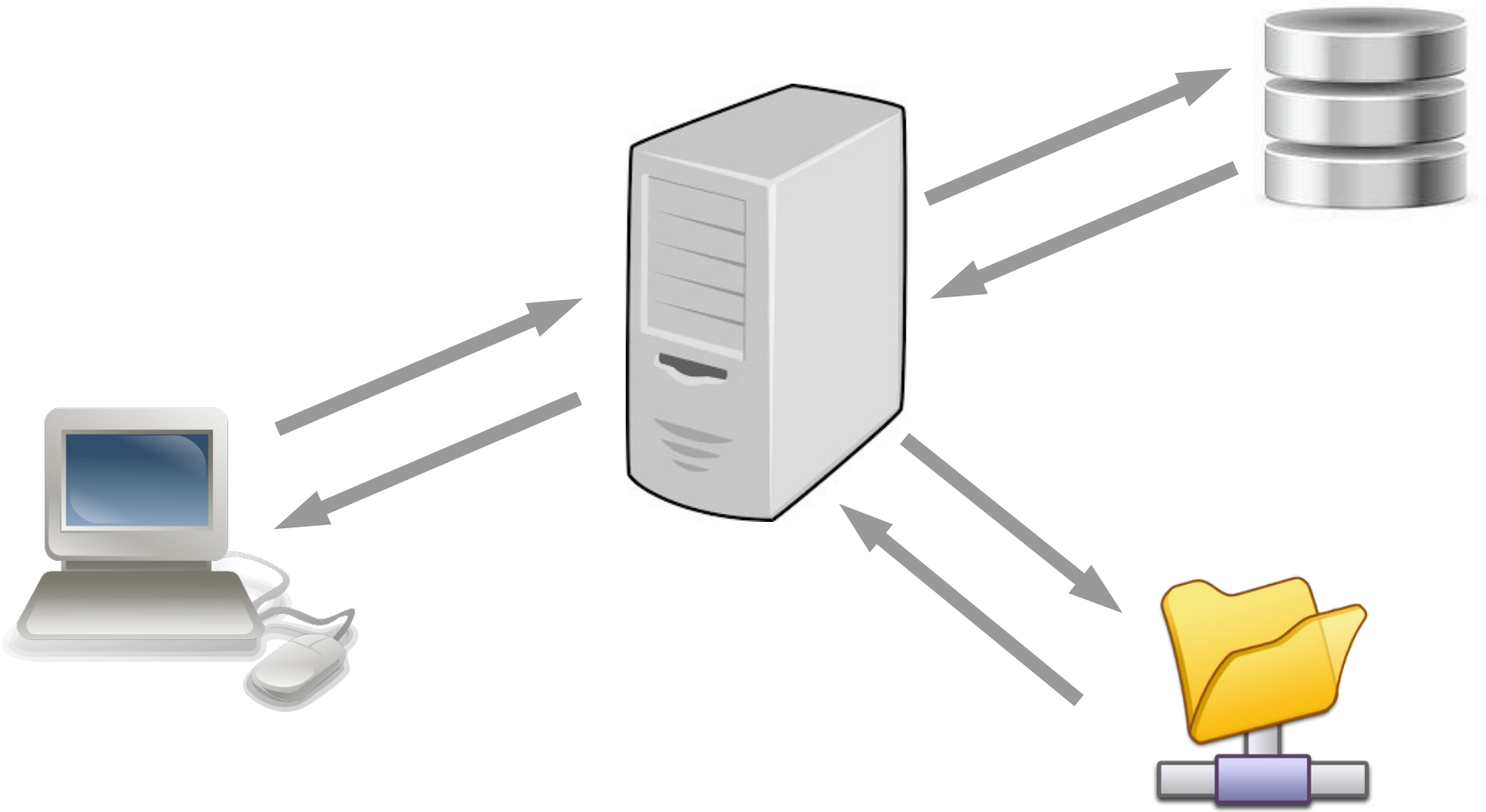
PHP: dati in arrivo dal client

*“Input validation is both the **most fundamental defense** that a web application relies upon and **the most unreliable**. A significant majority of web application vulnerabilities arise from a validation failure, so getting this part of our defenses right is essential.”*

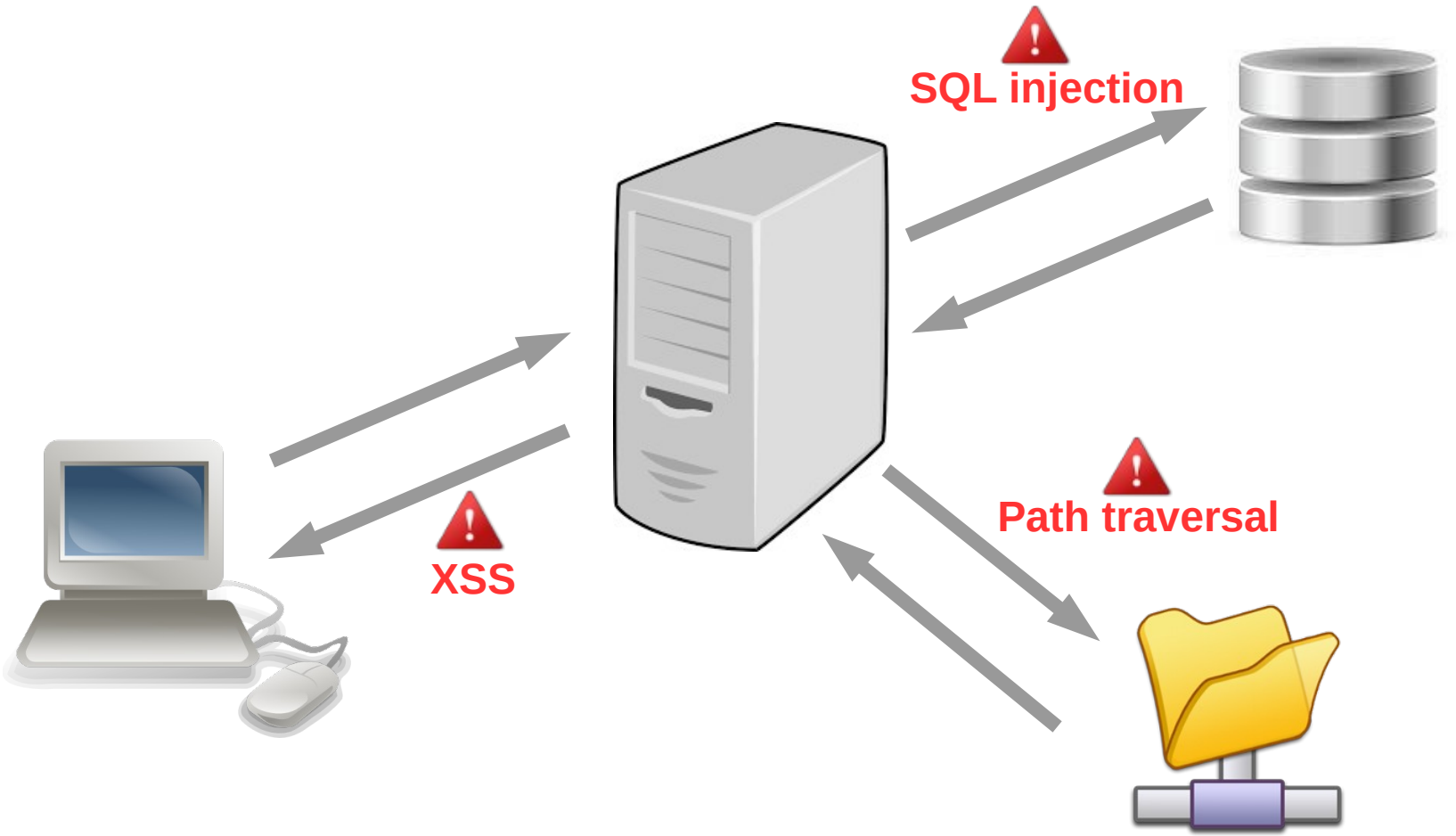
PHP: dati in arrivo dal client

*"For example, if I receive a **piece of data containing a name**, I may validate it fairly loosely to allow for apostrophes, commas, brackets, spaces, and the whole range of alphanumeric Unicode characters (not all of which need literally be alphabetic according to Western languages). As a name, **we'd have valid data which can be useful for display purposes** (it's first intended use). However, **if we use that data elsewhere** (e.g. a database query) we will be **putting it into a new context. In that new context, some of the characters we allow would still be dangerous** - our name might actually be a carefully crafted string intended to perform an SQL Injection attack."*

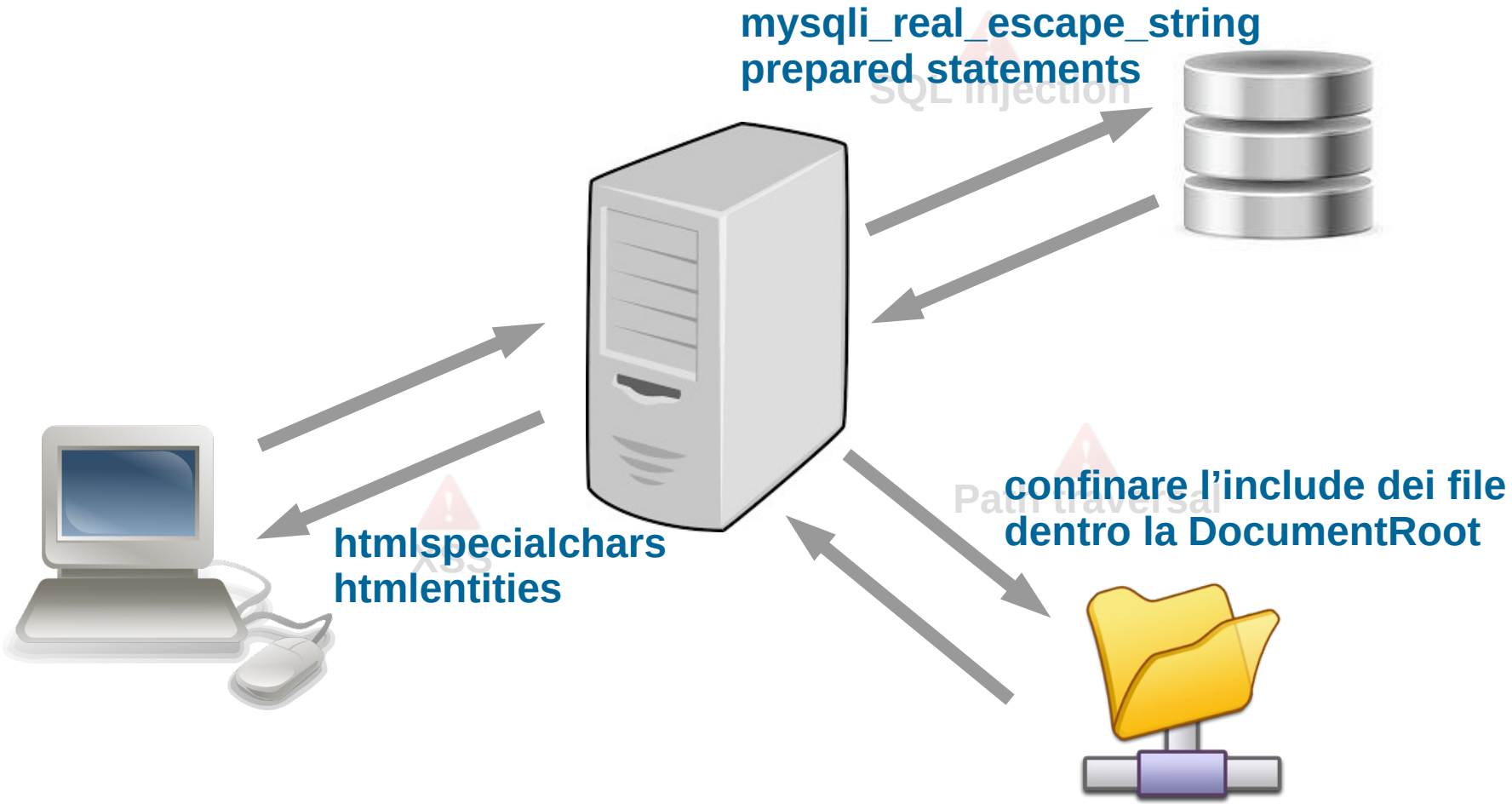
PHP: contesti di input/output



PHP: contesti di input/output



PHP: contesti di input/output



PHP: “utilities” input

isset($var)

- true (1), se $var esiste e non è NULL
- false (0) altrimenti

empty($var)

- true (1), se $var non esiste oppure il suo valore è false
- false (0), altrimenti

**non genera warning se la variabile non esiste*

PHP: “utilities” input

trim($str)

toglie spazi, \n, \r, \t, \v, \0 ad inizio e fine stringa

filter_var($str[, int $filter = FILTER_TYPE])

applica il filtro selezionato (regex) alla stringa

PHP: “utilities” output

htmlspecialchars($str, ENT_QUOTES[, $charset])

converte i caratteri speciali nelle entità HTML corrispondenti

PHP: parsing input

int($var) converte $var in intero, se possibile
(simile a int **intval**(mixed $var [, int $base = 10]))

Esempio: 12a -> 12, a12 -> 0

doubleval($var)

strval($var)

is_numeric($var) verifica se $var è un numero o una stringa

PHP: espressioni regolari

Si possono usare le **espressioni regolari** per definire dei **pattern** su stringhe di testo

PHP usa le espressioni regolari di Perl e offre delle **funzioni** per verificare se un certo pattern occorre in una stringa

Mediante l'uso di simboli come +,?,-,^,*,[ ],{ } si possono creare **espressioni complesse** per validare email, indirizzi IP, ecc.

PHP: espressioni regolari

Metacharacter	Description	Example
.	Matches any single character except a new line	/./ matches anything that has a single character
^	Matches the beginning of or string / excludes characters	/^PH/ matches any string that starts with PH
$	Matches pattern at the end of the string	/com$/ matches guru99.com,yahoo.com Etc.
*	Matches any zero (0) or more characters	/com*/ matches computer, communication etc.
+	Requires preceding character(s) appear at least once	/yah+oo/ matches yahoo
\	Used to escape meta characters	/yahoo+\.com/ treats the dot as a literal value
[...]	Character class	/[abc]/ matches abc
a-z	Matches lower case letters	/a-z/ matches cool, happy etc.
A-Z	Matches upper case letters	/A-Z/ matches WHAT, HOW, WHY etc.
0-9	Matches any number between 0 and 9	/0-4/ matches 0,1,2,3,4

The above list only gives the most commonly used metacharacters in regular expressions.

PHP: espressioni regolari

Esempi

/[0-9]{8,14}/

/[a-z\s]{10,20}/

/[a-zA-Z0-9\s]{1,}/

/^.{2}$/

PHP: espressioni regolari

Esempi

/[0-9]{8,14}/

/[a-z\s]{10,20}/

/[a-zA-Z0-9\s]{1,}/

/^.{2}$/

```
<?php
$str = "Ciao ciao ";
$pattern = "/^[a-z\s]{10,20}$/";
echo preg_match($pattern, $str);
?>
```

PHP: espressioni regolari

Esempi

/[0-9]{8,14}/

/[a-z\s]{10,20}/

/[a-zA-Z0-9\s]{1,}/

/^.{2}$/

```
<?php
$str = "Ciao ciao ";
$pattern = "/[0-9a-zA-Z\s]{10,20}/";
echo preg_match($pattern, $str);
?>
```