Imperial College London

Network and Web Security

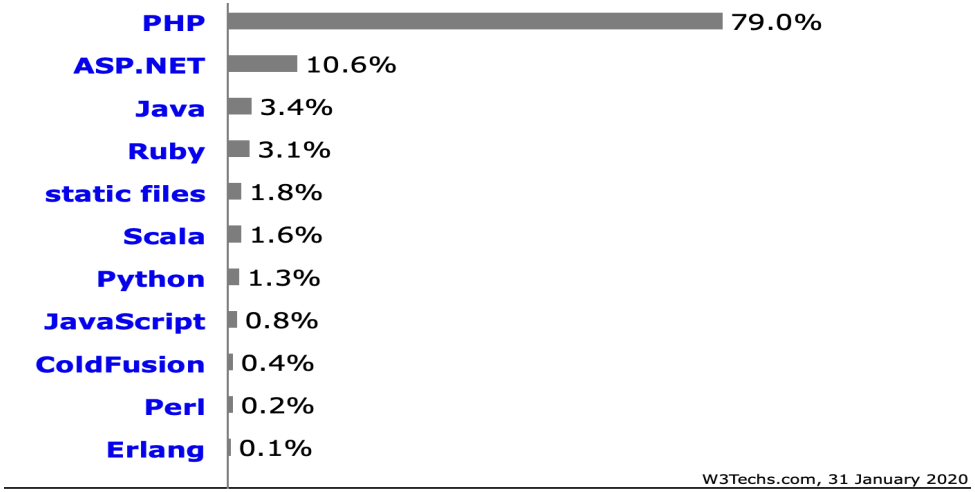
PHP

Dr Sergio Maffeis
Department of Computing
Course web page: https://331.cybersec.fun

Why PHP?

Imperial College London

- PHP is the predominant server-side language
  - Many large websites
    - Facebook
    - Baidu, Yahoo
    - Wikipedia, Wordpress
    - Pornhub 😐
    - …
  - Very large percentage of small websites
- Simple and practical
  - Fast development cycle
  - Easy to get started and to deploy
- Powerful and dangerous
  - Easy to make mistakes
    - Many practical examples of server-side vulnerabilities are on PHP
  - Preferred by attackers
    - Most exploit/phishing kits are written in PHP
- **Goal: understand enough PHP to read examples, find vulnerabilities, propose fixes**
  - Non-goal: become a proficient PHP programmer
  - Recommended exercise
    - Write a simple web app in PHP that can store HTTP POST data in a Postgres SQL or SQLite database



NVD

Search for CVEs, 5/2/21, for last 3 years

PHP	5,145
Java	2,452
Ruby	173
ASP.net	37

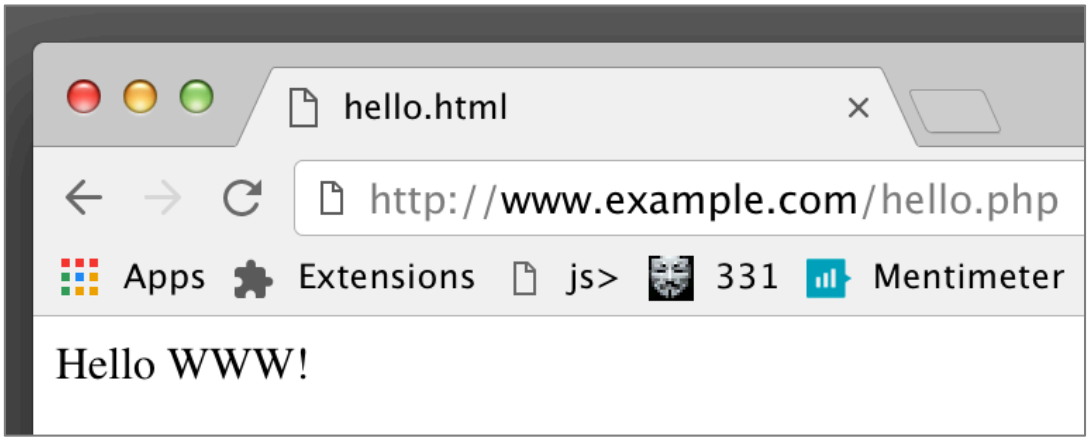
PHP by examples

Imperial College London

- Hello WWW
  - Client sends GET request `http://www.example.com/hello.php?name=WWW`
  - Server runs the PHP script

```
<? echo "<HTML><Body>Hello␣".$_GET["name"]."!</Body></HTML>"; ?>
```

  - Client receives personalised web page



PHP by examples

Imperial College London

- Imperative language with aliasing

```
$x = 0;
$y = &$x;          // $x and $y are now aliased
$y = "Hello!";
echo $x;           // prints "Hello!"
```

- Dynamic variable names

```
$x = "y";
$y = "Hello!";
echo $$x;                    // prints "Hello!"
```

  - Alternative notation, object access:

```
${"x"} = "y";
$z -> {"x".$x};
```

PHP by examples

Imperial College London

- Implicit type conversions (“*type juggling*”)

```
php> if (0) {echo "yes";} else {echo "no";}      // "no"
php> if ("0") {echo "yes";} else {echo "no";}    // "no"
php> if (0.0) {echo "yes";} else {echo "no";}    // "no"
php> if ("0.0") {echo "yes";} else {echo "no";} // "yes"
```

  - “0”=> false, “sssss”=> true (as boolean)

```
php> var_dump(3.2*"hi" + 45 - "3bye"*true); // float(42)
```

  - “sssss” => 0, “*nnn*ssss”=> *nnn* (as number)

```
php> var_dump("10"<"9");              // bool(false)
php> var_dump("10LOW"<"9HIGH");       // bool(true)
```

  - Crazy rules for comparison operator ($<_n$ vs $<_s$)
  - See: *Abstract domains for type juggling* (Arceri, Maffeis), 2016

PHP by examples

Imperial College London

- Arrays

```php
$x = array("foo" => "bar",4.5 => "baz");
$x[] = "default"; // use default key 5
echo $x[5];       // prints "default"
echo current($x); // prints "bar"
next($x);         // advances the pointer
echo current($x); // prints "baz"
```

- Objects

```php
$obj -> x = 0;
var_dump($obj);
> object(stdClass)#1 (1) { ["x"]=> int(0) }
```

```php
class par {
  private $id = "foo";
  function displayMe() {
    echo $this -> id; }}
```

```php
class chld extends par {
  public $id = "bar";
  public function displayHim() {
    parent::displayMe(); }}
```

```php
$obj = new chld();
$obj -> displayHim();    // prints "foo"
```

PHP by examples

Imperial College London

- Arrays view of environments

```
$GLOBALS["x"] = 42;
echo $x;                    // prints 42
```

- Subtle array-copy semantics

```
$x = array(1, 2, 3);
$y = $x;
$x[0] = "updated";
echo $y[0];                 // prints 1

$x = array(1, 2, 3);
$temp = &$x[1];             // we introduce sharing
$y = $x;                    // and assign normally
$x[0] = "regular";          // update a regular element
$x[1] = "shared";           // update the shared element
```

```
var_dump($x);
> array(3) {
    [0]=> string(7) "regular"
    [1]=> &string(6) "shared"
    [2]=> int(3) }
```

```
var_dump($y);
> array(3) {
    [0]=> int(1)
    [1]=> &string(6) "shared"
    [2]=> int(3) }
```

PHP by examples

Imperial College London

- Functions, and delayed reference resolution

```
function mod_x() {
        global $x;
        $x = array('a','b');
        return 0;
}

$x = array(1, 2);
$x[0] = mod_x();
var_dump($x);

>array(2) { [0]=> int(0)
            [1]=> string(1) "b" }
```

Analysis of PHP

Imperial College London

- PHP is hard to analyse statically
  - Interplay of aliasing, objects, type conversions, dynamic string-to-code conversion, copy-on-write optimisation
  - Hack (https://hacklang.org) restricts PHP to provide static type system
- Practical PHP analysis tools (Fortify, Pixy, Checkmarx…)
  - Mostly based on taint analysis and string analysis
  - Coarse over/under-approximation to limit false positives/negatives
- Research on static analysis
  - Dahse, Holz: *Static Detection of Second-Order Vulnerabilities in Web Applications*. USENIX Security 2014
  - Hauzar, Kofron: *Framework for Static Analysis of PHP Applications*. ECOOP 2015
  - Filaretti: *An executable formal semantics of PHP with applications to program analysis.* DOC PhD Thesis, 2016
  - Backes, et al: *Efficient and Flexible Discovery of PHP Application Vulnerabilities*. EURO S&P 2017
- DeepTective: prize winning DOC MEng project
  - Rabheru, Hanif, Maffeis : *A Hybrid Graph Neural Network Approach for Detecting PHP Vulnerabilities*. ArXiv 2020 and SAC 2021
  - Found 4 new vulns in Wordpress plugins, 2 CVEs