

HTTP Protocol

Goal

- Present the history of the HTTP
- Show the key features of the protocol
- Present the definition of Web Security and give a classification of the attacks
- List useful tools commonly used in Web Security

Outline

- HTTP History
- Key Features and Overview of HTTP
- Security and Web Security
- Tooling

HTTP History

- HTTP was introduced at the beginning of the '90s
- The first version of the protocol, **HTTP 0.9**, was released under the World Wide Web initiative
 - Extremely simple
 - Released in 1991
- <https://www.w3.org/Protocols/HTTP/AsImplemented.html>

HTTP History

- HTTP is
- Every c
 - The f
 - was t
- The first
- "WorldWide"
- It was r
 - <http://>

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

[What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) ,X11 [Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#))

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help ?](#)

If you would like to support the web..

[Getting code](#)

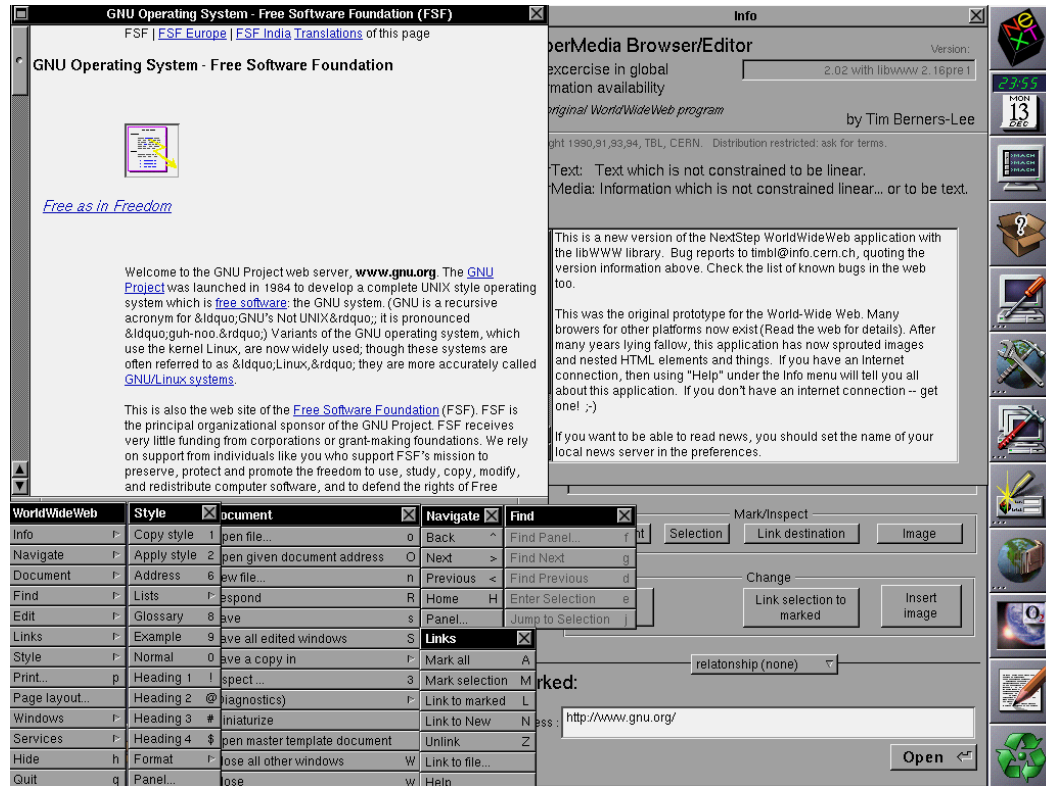
Getting the code by [anonymous FTP](#) , etc.

se main goal

;

HTTP History

- HTTP is the first
- Every document has a URL
- The first document was the first
- The first "WorldWideWeb"



the main goal

HTTP - History

- Through the years, **browsers became more and more complex**
- **Mosaic** in mid-1993 brought new features, such as the possibility to embed images into web pages
- Several software houses started to develop their own browser, adding new features to defeat the concurrence
 - HTML enchantments
 - JavaScript
 - Plugins such as Java/Flash

HTTP - History

- This race led to a vast diversity of standards
- Each browser implemented its own heuristics to maintain compatibility with other browsers
 - Often ignoring all the security implications

HTTP Overview - URLencoding

- Looking at the URL syntax one may notice that some characters have a **special meaning** in a URL
 - The character “#” is used for segments, and the server will always ignore every character after it
 - The character “&” is used as a variable separator in the URL-Path
 - ...

HTTP Overview - URLencoding

- What if we want to send the text “hello &# world” in a GET variable?
- Because the fragment is reserved for clients, the server will **ignore** the word "world"

`http://foobar.com/?var=hello &# world`

HTTP Overview - URLEncoding

- This problem is solved using a **particular encoding**, that converts every character in a “not harmful” representation
- This encoding is called “**URL encoding**” or “**Percent Encoding**”

HTTP Overview - URLEncoding

- This encoding is very simple
 - Take the hex value of a character you want to encode, and then prepend a “%” symbol
== %23
- Every reserved character in a URL must be urlencoded
- Every non-printable character must be urlencoded
- Spaces can be represented either with %20, or with the plus sign (+)

HTTP Overview - URLEncoding

- So, the following not valid URL:

`http://foobar.com/?var=hello &# world`

- Is rewrote as:

`http://foobar.com/?var=hello+%26%23+world`

HTTP Overview

- **Requests** and **Responses** are HTTP messages composed of three different parts
 - The Request/Response line
 - The Header Fields
 - A Body (optional)
- Every line in the Request/Response is terminated by the “CR;LF” sequence: “\r\n” or 0x0d0a in binary
- An empty line separates the last header field from the body

HTTP Overview - Requests



HTTP Overview - Requests

- The Request line is composed of
 - A **method**
 - What you want to do with the resource
 - The **resource** for which we are doing the request
 - Contains the URL-Path of the
 - And, finally, the **protocol version**
 - The client and the server must agree on the protocol version

GET / HTTP/1.1

HTTP Overview - Methods

- Tell the server “what we are doing” with the resource
- Most used methods
 - GET
 - POST
 - OPTIONS
 - HEAD
 - PUT
 - DELETE
- There are a lot of less used methods
 - CONNECT
 - TRACE
 - PATCH
 - etc...

HTTP Overview - Headers

- Headers are used to send **additional data** to the server
- Serialized in the form **name: value**
- Some are mandatory:
 - **host**
 - **content-encoding/content-length** if there is a body
- Most used headers are:
 - **cookie**
 - **referer**
 - **user-agent**

HTTP Overview - Body

- **Generic data** sent to the server
- Its type (or encoding) is defined by the **Content-Type** header
- It can be **encoded** in different ways:
 - application/x-www-form-urlencoded
 - text/plain
 - application/json
 - ...
- It can also have a **custom encoding**:
 - foo/bar
 - ...

HTTP Overview - Responses

- A Response is very similar to a Request
- It differs only for the **first line**, which is called “status-line”
- This line is mandatory, and tells the client the type of the response and the version of the protocol used to make the response

HTTP Overview - Responses

```
HTTP/1.1 200 Ok
```

```
Host: 127.0.0.1:5000
```

```
Date: Thu, 12 Mar 2020 10:31:38 GMT
```

```
Connection: close
```

```
X-Powered-By: PHP/7.4.3
```

```
<b>Hello World!</b>
```

Status-Line

Header Fields

Body Field

HTTP Overview – Status Line

- The status-line composed by the **version of the protocol**, an **integer number**, and a **string**
- The number is called **status code**. Status codes are divided into five categories:
 - 1**: Informational Response
 - 2**: Success
 - 3**: Location change
 - 4**: Client Error
 - 5**: Server Error

HTTP Overview – Status code

- Some common status codes are (informal way)
 - 1xx: hold on
 - 2xx: here you go
 - 3xx: go away
 - 4xx: you fucked up
 - 5xx: I fucked up

HTTP Overview – Status code

- Some common status codes are
 - 200: The request was successful
 - 400: The request was malformed
 - 404: The requested resource could not be found
 - 500: The server had a critical error, and could not complete the request

HTTP Overview - Cookies

- In order to make HTTP stateful, **cookies** were introduced
- Cookies are **text information** that a web client receives and stores from a server, and sends back within every request to the host
- They are used mainly for
 - Session management
 - Personalization
 - Tracking



HTTP Overview - Cookies

- HTTP servers can set cookies with the response header field **Set-Cookie**
- Cookies can also be set client-side via JavaScript
- Cookies are composed by a name, a value, and some meta-information
 - The origin (e.g., the server which sends the cookie)
 - The expire date
 - Some security policies

HTTP Overview - Cookies

- Browsers will send back cookies to the server in accordance with its scope
- The scope is the “origin” in which each cookie was created
 - If a cookie named “foo” is set by “www.google.com”, it cannot be sent to “www.microsoft.com”, but only to “www.google.com”

HTTP Overview

Hands on a real webpage.
Go to <https://training.olicyber.it/>

Does this website sets cookies?

Does this website sets cookies?

No!

Now login. What changes?

Now login. What changes?

There is a cookie set

Is this cookie related to the login?

Is this cookie related to the login?

No!

How is the login handled?

How is the login handled?

How does the website recognise us?

How is the login handled?

How does the website recognise us?

Is there another useful header?

How is the login handled?

How does the website recognise us?

Is there another useful header?

The login is handled using an “authorization” header. It contains a token that allows the server to recognise us.

Introduction to PHP

About the PHP Language

- Syntax inspired by C
 - Curly braces, semicolons, no significant whitespace
- Syntax inspired by perl
 - Dollar signs to start variable names, associative arrays
- Extends HTML to add segments of PHP within an HTML file

Philosophy of PHP

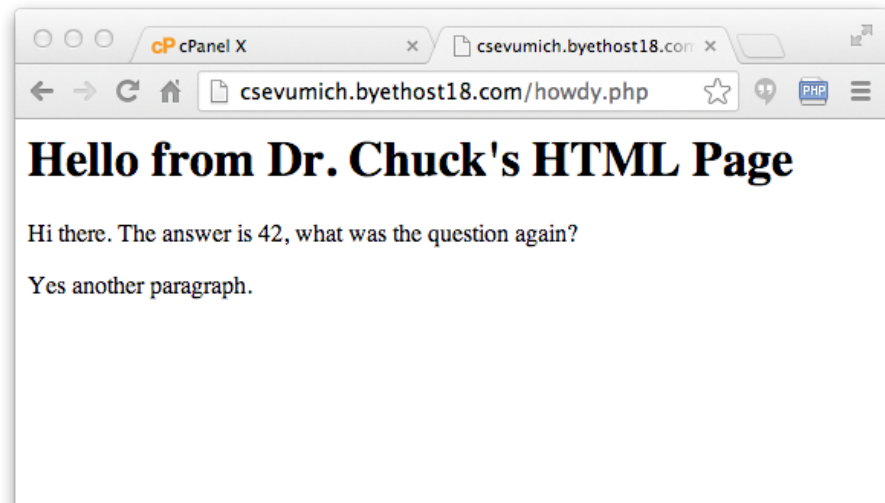
- You are a responsible and intelligent programmer.
- You know what you want to do.
- Some flexibility in syntax is OK - style choices are OK.
- Let's make this as convenient as possible.
- Sometimes errors fail silently.

Example of a PHP page

```
<h1>Hello from Dr. Chuck's HTML Page</h1>
<p>
  <?php
    echo "Hi there.\n";
    $answer = 6 * 7;
    echo "The answer is $answer, what ";
    echo "was the question again?\n";
  ?>
</p>
<p>Yes, another paragraph.</p>
```

Example of a PHP page

```
<h1>Hello from Dr. Chuck's HTML Page</h1>
<p>
    <?php
        echo "Hi there.\n";
        $answer = 6 * 7;
        echo "The answer is $answer, what ";
        echo "was the question again?\n";
    ?>
</p>
<p>Yes, another paragraph.</p>
```



Keywords

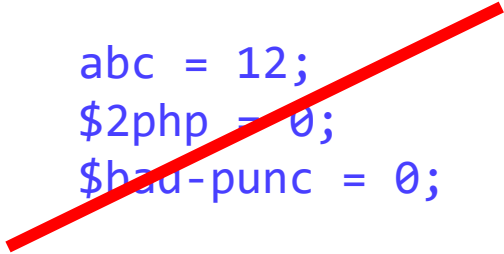
abstract and array() as break case catch class clone const continue
declare default do else elseif end declare endfor endforeach endif
endswitch endwhile extends final for foreach function global goto if
implements interface instanceof namespace new or private protected
public static switch \$this throw try use var while xor

Variable Names

- Start with a dollar sign (\$) followed by a letter or underscore, followed by any number of letters, numbers, or underscores
- Case matters

```
$abc = 12;  
$total = 0;  
$largest_so_far = 0;
```

```
abc = 12;  
$2php = 0;  
$had-punc = 0;
```



Variable Name Weirdness

- Things that look like variables but are missing a dollar sign as an array index could produce strange results...

```
$x = 5;  
$y = array("x" => "Hello");  
print $y[x];
```

Variable Name Weirdness

- Things that look like variables but are missing a dollar sign as an array index could produce strange results...

```
$x = 5;  
$y = array("x" => "Hello");  
print $y[x];
```

Hello

Variable Types

- PHP is not a strong typed language
- Each variable has a type but it is inferred from the interpreter
- PHP also supports objects like OO programming languages
- To print the whole content of a variable use the `var_dump` function

Strings

- String literals can use single quotes or double quotes
- The backslash (\) is used as an “escape” character
- Strings can span multiple lines
 - the newline is part of the string
- In double-quoted strings, variable values are expanded
- Concatenation is the "." not "+"

Strings - Example

```
<?php  
echo "this is a simple string\n";  
?>
```

Strings - Example

```
<?php  
echo "this is a simple string\n";  
?>
```

this is a simple string

Strings - Example

```
<?php  
echo "You can also have embedded newlines in  
strings this way as it is  
okay to do";  
?>
```

Strings - Example

```
<?php  
echo "You can also have embedded newlines in  
strings this way as it is  
okay to do";  
?>
```

You can also have embedded newlines in
strings this way as it is
okay to do

Strings - Example

```
<?php  
echo "This will expand: \na newline";  
?>
```

Strings - Example

```
<?php  
echo "This will expand: \na newline";  
?>
```

This will expand:
a newline

Strings - Example

```
<?php  
$expand = 12;  
echo "Variables do $expand\n";  
?>
```

Strings - Example

```
<?php  
$expand = 12;  
echo "Variables do $expand\n";  
?>
```

Variables do 12

Strings - Example

```
<?php  
echo 'Arnold once said: "I\'ll be back"';  
?>
```

Strings - Example

```
<?php  
echo 'Arnold once said: "I\'ll be back"';  
?>
```

Arnold once said: "I'll be back"

Strings - Example

```
<?php  
echo 'This will not expand: \n a newline';  
?>
```

Strings - Example

```
<?php  
echo 'This will not expand: \n a newline';  
?>
```

This will not expand: \n a newline

Strings - Example

```
<?php  
$expand = 12;  
$either = "ciao";  
echo 'Variables do not $expand $either';  
?>
```

Strings - Example

```
<?php  
$expand = 12;  
$either = "ciao";  
echo 'Variables do not $expand $either';  
?>
```

Variables do not \$expand \$either

Comments in PHP

```
echo 'This is a test'; // This is a c++ style comment
/* This is a multi line comment
yet another line of comment */
echo 'This is yet another test';
echo 'One Final Test'; # This is a shell-style comment
```

Expressions

- Completely normal like other languages (+ - / *)
- More aggressive implicit type conversion

```
<?php
    $x = "15" + 27;
    echo($x);
    echo("\n");
?>
```

Expressions

- Completely normal like other languages (+ - / *)
- More aggressive implicit type conversion

```
<?php
    $x = "15" + 27;
    echo($x);
    echo("\n");
?>
```

42

Operators

- Increment / Decrement (++ --)
- String concatenation (.)
- Equality (== !=)
- Identity (=== !==)
- Ternary (? :)
- Side-effect Assignment (+= -= .= etc.)
- Ignore the rarely-used bitwise operators (>> << ^ | &)

String Concatenation

- PHP uses the period character for concatenation, because the plus character would instruct PHP to do the best it could to add the two things together, converting if necessary.

```
$a = 'Hello ' . 'World!';  
echo $a . "\n";
```

String Concatenation

- PHP uses the period character for concatenation, because the plus character would instruct PHP to do the best it could to add the two things together, converting if necessary.

```
$a = 'Hello ' . 'World!';  
echo $a . "\n";
```

Hello World!

Conversion / Casting

- As PHP evaluates expressions, sometimes values in the expression need to be converted from one type to another as the computations are done.
 - PHP does aggressive implicit type conversion (casting).
 - You can also make type conversion (casting) explicit with casting operators.

Casting

```
$a = 56; $b = 12;  
$c = $a / $b;  
echo "C: $c\n";  
$d = "100" + 36.25 + TRUE;  
echo "D: ". $d . "\n";  
echo "D2: ". (string) $d . "\n";  
$e = (int) 9.9 - 1;  
echo "E: $e\n";  
$f = "sam" + 25;  
echo "F: $f\n";  
$g = "sam" . 25;  
echo "G: $g\n";
```

Casting

```
$a = 56; $b = 12;  
$c = $a / $b;  
echo "C: $c\n";  
$d = "100" + 36.25 + TRUE;  
echo "D: ". $d . "\n";  
echo "D2: ". (string) $d . "\n";  
$e = (int) 9.9 - 1;  
echo "E: $e\n";  
$f = "sam" + 25;  
echo "F: $f\n";  
$g = "sam" . 25;  
echo "G: $g\n";
```

```
C: 4.666666666667  
D: 137.25  
D2: 137.25  
E: 8  
F: 25  
G: sam25
```

Equality versus Identity

- The equality operator (==) in PHP is far more aggressive than in most other languages when it comes to data conversion during expression evaluation.

```
if ( 123 == "123" ) print ("Equality 1\n");  
if ( 123 == "100" + 23 ) print ("Equality 2\n");  
if ( FALSE == "0" ) print ("Equality 3\n");  
if ( (5 < 6) == "2" - "1" ) print ("Equality 4\n");  
if ( (5 < 6) === TRUE ) print ("Equality 5\n");
```

Python Requests

Learning with practise

Do a GET request

- Create a python script that uses the requests library
- Make it do a GET request to `web-01.challs.olicityber.it`

GET Request with parameters

- Create a python script that uses the requests library
- Obtain the flag by doing a GET request on
`http://web-02.challs.oolicyber.it/server-records`
- with parameter "id" and value "flag"

Custom Header

- Create a python script that uses the requests library
- Obtain the flag by doing a GET request on
`http://web-03.challs.oolicyber.it/flag`
- with header "X-Password" containing "admin" as value

Header Accept

- Create a python script that uses the requests library
- Obtain the flag by doing a GET request on
`http://web-04.challs.olicityber.it/users`
- allowing the server only to give you the "application/xml" format

POST Request

- Create a python script that uses the requests library
- Obtain the flag by doing a POST request on
`http://web-08.challs.olicityber.it/login`
- with an HTTP form (application/x-www-form-urlencoded type) containing
 - `username: admin`
 - `password: admin`

POST Request – JSON format

- Create a python script that uses the requests library
- Obtain the flag by doing a POST request on
`http://web-09.challs.olicityber.it/login`
- sending in JSON the values
 - `username: admin`
 - `password: admin`

Web Security

Web Security

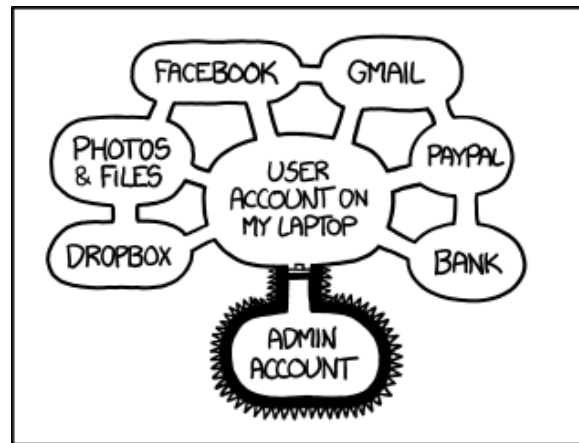
- Web security applies to vulnerabilities that affect web applications.
- Typically, web applications are the most exposed assets to an attacker
- And http is really fragile..

Web Security - History

- HTTP was created with the intent to serve **static documents**
- The protocol is **simple by design**
 - It is a **stateless** protocol, since there was no need to keep track of the current client
 - **Documents were simple**, there was no need for animations
 - The **security was not a big concern**, at the beginning there was not much to protect on the web

Web Security

- But now we have
 - **Dynamic generated pages**, e.g., scripts that generate pages on-the-fly
 - Exceptionally **complex pages**: HTML CSS, JavaScript, WebAsm, plugins... and a lot more
 - A lot of **secrets to protect**
- Such complexity leads to a **huge attack surface**



IF SOMEONE STEALS MY LAPTOP WHILE I'M LOGGED IN, THEY CAN READ MY EMAIL, TAKE MY MONEY, AND IMPERSONATE ME TO MY FRIENDS, BUT AT LEAST THEY CAN'T INSTALL DRIVERS WITHOUT MY PERMISSION.

The web is a mess...

Web Security

- Web security is about the security of web assets, e.g., everything that runs over HTTP
 - **Server-Side Security:** The impact affects the remote server
 - **Client-Side Security:** The impact affects the client
 - Note: This does not mean that it is a vulnerability of the browser!
 - Rule of thumb: *If you need to send a link to the victim, then probably it is a client-side vulnerability*

Web Security – Some useful tools

- Browser
 - With the Developer Tools
- Curl/wget
 - A Command line utility to make http requests
- Python requests
 - A useful python library to do http requests
- ~~■ Burp suite/zap proxy~~
 - ~~• live edit raw http requests and response~~
- Test server (php dev & httpsimplepython)
- webhook.site
 - Simple webserver that logs everything

On-the-fly PHP HTTP server

- A very fast-to deploy test server
- Serves every file inside the directory it was launched from and executes .php scripts
- `php -S 127.0.0.1:5000`
 - Launch it from a test directory! You don't want to leak your .ssh directory !