# WS 1.1 - HTTP Protocol And Web Security Overview

—

Polonium - Pwnzer0tt1

gh repo fork WS_1.1 - HTTP Protocol and Web-Security Overview

# Prerequisites

- NS 0.1 - Network Fundamentals
- A computer
- How to use a search engine
- ~~Rust~~ JavaScript

# Outline

- Key Features and Overview of HTTP
- Tooling

# Key Features and Overview of HTTP

# Brief overview of the history of HTTP

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Evolution_of_HTTP

**1989**

**Invention of the World Wide Web**

While working at CERN, Tim Berners-Lee wrote a proposal to build a hypertext system over the internet.

**1990**

**Implementation**

Renamed the World Wide Web during its implementation in 1990. Built over the existing TCP and IP protocols, it consisted of 4 building blocks: HTML, HTTP, a client and a server.

**August 6, 1991**

**Go public**

Tim Berners-Lee posted on alt.hypertext newsgroup. This is considered to be the official start of the WWW as a public project.

**1996**

**HTTP/1.0**

In 1991 the first documented version was HTTP/0.9, which supported only GET method. RFC 1945 was published in May 1996 and was the final HTTP/1.0 revision.

**2000**

**HTTPS**

HTTPS was formally specified by RFC 2818, TLS replaced SSL.

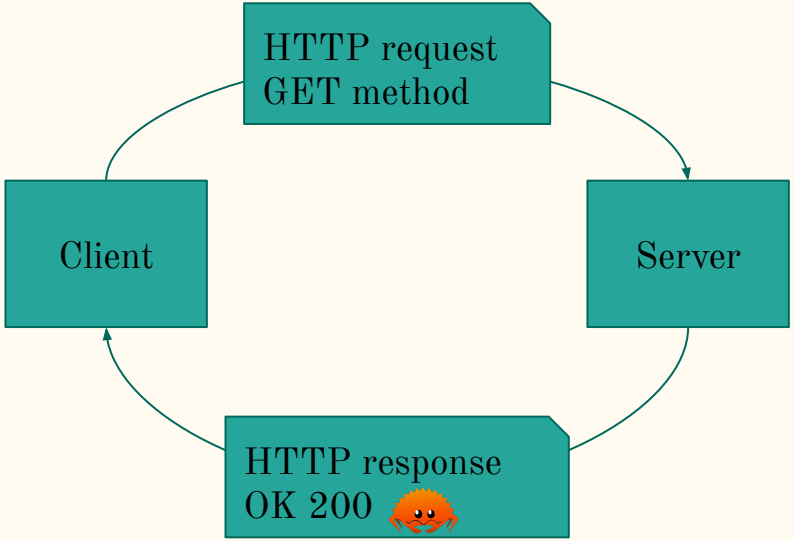
**20XX**

**Evolution**

Due to the widespread adoption of the World Wide Web many things have changed. Today the latest version is HTTP/3, UDP is used, TLS 1.0 is deprecated, and JavaScript has taken over the world.

# HTTP Overview

- HTTP/1.1 defined in RFC 2616
- Application layer in the ISO/OSI stack
- Based on TCP
- Human readable
- Client-Server architecture
- Stateless



# HTTP Overview

HTTP is used to transfer resources to a client that made a request.

A resource can be:

- HTML file
- Images, videos
- Text documents
- ...

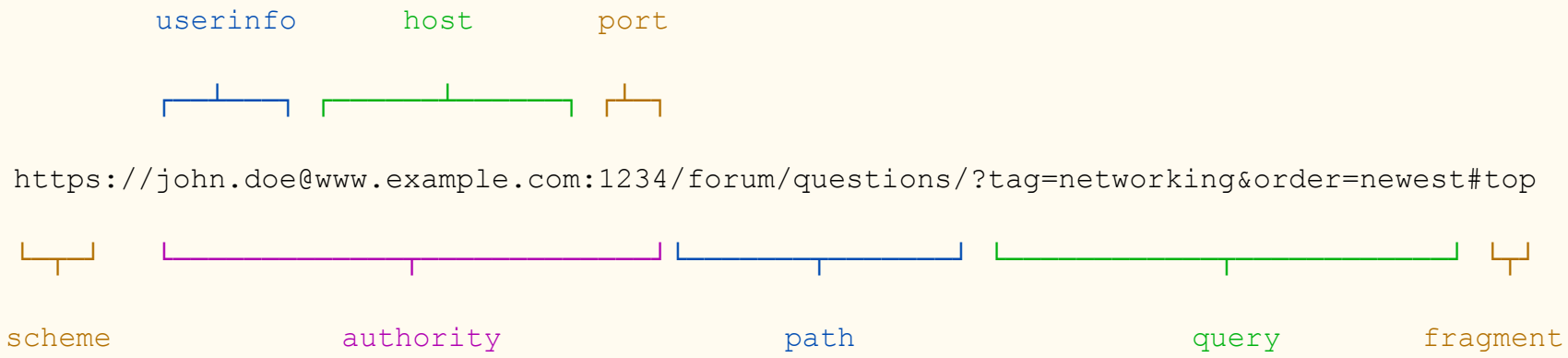
# HTTP Overview - URL

Source: https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL

A resource is uniquely represented with a URL (Uniform Resource Locator).

URLs are defined in RFC 1739 and are a type of URI (RFC 3968);

Example:



# HTTP Overview - URL

```
https://john.doe@www.example.com:1234/forum/questions/?tag=networking&order=newest#top
```

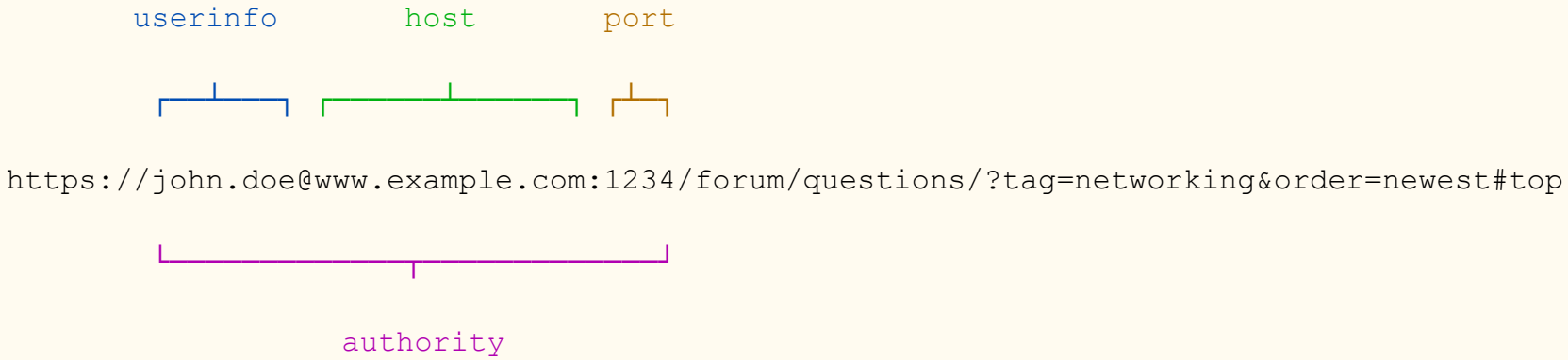
scheme

The **scheme** specifies the protocol used.

For websites the protocol is http or https (secured version).

Other schemes such as mailto, file or ftp exist.

# HTTP Overview - URL




~~**Userinfo** consist of a username and an optional password.~~ **Userinfo** is deprecated for security reasons.

The **host** (domain or IP) and **port** (80 or 443 by default) represent the address to which the client should send requests.

# HTTP Overview - URL

```
https://john.doe@www.example.com:1234/forum/questions/?tag=networking&order=newest#top
```

path

The **path** is the resource requested to the server.

It can be a “physical” or “virtual” location:

`/page.html`

`/user/32498232`

# HTTP Overview - URL

```
https://john.doe@www.example.com:1234/forum/questions/?tag=networking&order=newest#top
```

query

The **query** is optional and contains additional parameters provided to the server.

It is a list of key/value pairs separated by *&*. The values of the parameters must be encoded using the URLencoding.

# HTTP Overview - URLencoding

URL encoding (or percent-encoding), is a format used to encode data in a URI using ASCII characters.

Every reserved character in a URL must be encoded.

Every non-printable character must be encoded.

Example:

https://example.com?msg=Hello World

https://example.com?msg=Hello%20World

# HTTP Overview - URLencoding

There is a problem:

```
> console.log(new URL('http://exa mple.com').origin)
  http://exa%20mple.com
<· undefined
```

```
Welcome to Node.js v20.13.1.
Type ".help" for more information.
> console.log(new URL('http://exa mple.com').origin)
Uncaught TypeError: Invalid URL
    at new URL (node:internal/url:797:36) {
  code: 'ERR_INVALID_URL',
  input: 'http://exa mple.com'
}
```

# HTTP Overview - URL

```
https://john.doe@www.example.com:1234/forum/questions/?tag=networking&order=newest#top
```

fragment

The **fragment** is an optional component, client only, preceded by a hash #.

The fragment contains the fragment identifier that is used to point to a specific location inside the resource.

Example:

https://en.wikipedia.org/wiki/Rickrolling#History

# HTTP Overview

HTTP is a client-server protocol, this means requests are sent by the client and the server returns a response. The structure of requests and responses is similar:

Request

Start line

```
POST / HTTP/1.1
```

Headers

```
Host: example.com
User-Agent: curl/8.6.0
Accept: */*
Content-Type:
application/json
Content-Length: 23
```

Empty line

Body

```
{
"data": "Hello World!"
}
```

Response

```
HTTP/1.1 403 Forbidden
```

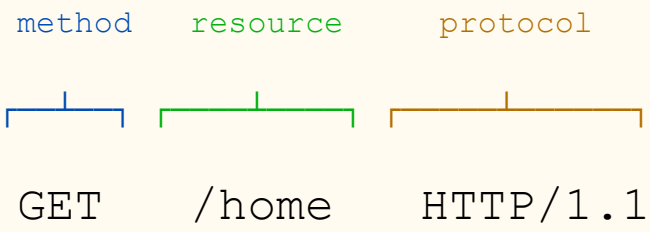
```
Server: Apache
Date: Fri, 21 Feb 2025
15:30:23 GMT
Content-Length: 678
Content-Type: text/html
Cache-Control: no-store
```

```
<!DOCTYPE html>
<html lang="en">
…
```

# HTTP Overview - Request line

The **request line** is composed of:

- HTTP method (GET, POST, etc...)
- Requested resource (absolute or relative URL)
- Protocol version (HTTP version to use)



# HTTP Overview - Request methods

Resource: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

An **HTTP method** is one of a set of defined words that describe the meaning of the request and the desired outcome. In particular:

- GET: the client would like to retrieve a resource;
- POST: the client is sending data to the server;
- DELETE: the client would like to delete the specified resource;
- HEAD: ask for a response to a GET request, but without a response body;
- PUT: replace a target resource with the representation the client gave;;
- OPTIONS, TRACE, PATH, CONNECT...

# HTTP Overview - Headers

**Headers** are metadata sent with a request after the start line and before the body.

Each header is a single line composed by a **case-insensitive** string followed by a colon : and a value whose format depends on the header. Some headers are specific to requests, others can be sent in both requests and responses.

| | |
|---|---|
| :authority: | www.rust-lang.org |
| :method: | GET |
| :path: | / |
| :scheme: | https |
| Accept: | text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 |
| Accept-Encoding: | gzip, deflate, br, zstd |
| Accept-Language: | it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7,fr;q=0.6 |
| Cache-Control: | max-age=0 |
| Priority: | u=0, i |
| Sec-Ch-Ua: | "Not A(Brand";v="8", "Chromium";v="132", "Google Chrome";v="132" |
| Sec-Ch-Ua-Mobile: | ?0 |
| Sec-Ch-Ua-Platform: | "Windows" |
| Sec-Fetch-Dest: | document |
| Sec-Fetch-Mode: | navigate |
| Sec-Fetch-Site: | none |
| Sec-Fetch-User: | ?1 |
| Upgrade-Insecure-Requests: | 1 |
| User-Agent: | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36 |

# HTTP Overview - Response line

The start line is called a **status line** in responses, and is composed by:

- Protocol (HTTP version)
- Status code (200, 302, 404, 500...)
- Status text (textual description of the status code)

# HTTP Overview - Status codes

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP status codes indicate the outcome of the request. Responses are grouped in five classes:

- Information responses (100-199)
- Successful responses (200-299)
- Redirection messages (300-399)
- Client error responses (400-499)
- Server error responses (500-599)

| | |
|---|---|
| Content-Encoding: | gzip |
| Content-Security-Policy: | default-src 'self'; frame-ancestors 'self'; img-src 'self' avatars.githubusercontent.com; frame-src 'self' player.vimeo.com |
| Content-Type: | text/html; charset=utf-8 |
| Date: | Wed, 05 Feb 2025 19:04:30 GMT |
| Nel: | {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_headers":["Via"]} |
| Permissions-Policy: | interest-cohort=() |
| Referrer-Policy: | no-referrer, strict-origin-when-cross-origin |
| Report-To: | {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1738782270&sid=67ff5de4-ad2b-4112-9289-cf96be89efed&s=Ci5uyzqjSN2smKUadderhfpIYevVA%2BfQEUXTKCM%2B0bw%3D"}]} |
| Reporting-Endpoints: | heroku-nel=https://nel.heroku.com/reports?ts=1738782270&sid=67ff5de4-ad2b-4112-9289-cf96be89efed&s=Ci5uyzqjSN2smKUadderhfpIYevVA%2BfQEUXTKCM%2B0bw%3D |
| Server: | Rocket |
| Strict-Transport-Security: | max-age=63072000 |
| Vary: | accept-encoding |
| Via: | 1.1 vegur, 1.1 46276a4e83fa64f4e593c2be8065a82a.cloudfront.net (CloudFront) |
| X-Amz-Cf-Id: | 1MJOHcscb16tP6kwSuLlt_YqKqbXeyMR1ffukqFld4bqleNpbhBJgA== |
| X-Amz-Cf-Pop: | FCO50-P5 |
| X-Cache: | Miss from cloudfront |
| X-Content-Type-Options: | nosniff |
| X-Frame-Options: | SAMEORIGIN |
| X-Xss-Protection: | 1; mode=block |

# HTTP Overview - Cookies

HTTP is stateless, this mean the server doesn't know if two different requests came from the same client. This is a problem especially in the case of resources that need authentication in order to be accessed.

A **cookie** is a small piece of data a server sends to a client. The client may store or modify cookies, create a new one, and send them back to the same server with later requests.

Cookies allows web applications to store limited amounts of data and remember states.

# HTTP Overview - Cookies

A cookie is set by a server using the field **Set-Cookie** (a server can set multiple Set-Cookie headers, each one for a separate cookie) inside the headers section of a response. The format contains a list of key/value separated by a semicolon “;”.

The attributes are:

- `<cookie-name>=<cookie-value>`
- `Domain=<domain-value>*`
- `Expires=<date>*`
- `HttpOnly*`
- `Max-Age=<number>*`
- `Path=<path-value>*`
- `SameSite=<samesite-value>*`
- `Secure*`
- `Partitioned*`

*Optional

# HTTP Overview - Cookies

The first attribute is the cookie name and its value.

*cookie-name* can contain ASCII characters except for control characters (0 up to 31 and 127) and separator characters *( ) < > @ , ; : \ " / [ ] ? = { }*.

*cookie-value* can optionally be wrapped in double quotes and include any ASCII character excluding: control characters, whitespace, double quotes, commas, semicolons and backslashes.

```
session=e32adc1d09ebc
```

# HTTP Overview - Cookies

**Domain** defines the host to which the cookie will be sent.

If set, the cookie will be available to the specified domain and all its subdomains.

```
Domain=example.com
```

It means is available to: *example.com*, *shop.example.com* and others subdomains.

# HTTP Overview - Cookies

**Expires** indicates the maximum lifetime of the cookie. When the set date is reached the cookie is considered expired.

The date is relative to the client the cookie is being set on.

If not specified a cookie will be treated as a **session cookie**, when the client shuts down the cookie is removed.

```
Expires=Wed, 21 Oct 2015 07:28:00 GMT
```

# HTTP Overview - Cookies

**HttpOnly** if present, forbids JavaScript from accessing the cookie (e.g. *Document.cookie*).

**Max-Age** indicates the number of seconds until the cookie expires. A zero or negative number will expire the cookie immediately.

If both **Expires** and **Max-Age** are set, **Max-Age** has precedence.

# HTTP Overview - Cookies

**Path** indicates the path that must exist in the requested URL for the browser to send the **Cookie** header.

*Path=/*

*Path=/docs*

# HTTP Overview - Cookies

**Samesite** controls if a cookie is sent with cross-site requests (requests originating from a different domain or scheme), providing protection against CSRF (cross-site request forgery) attacks.

The possible values are:

- **Strict** means the browser sends the cookie only for same-site requests.
- **Lax (default)** means the cookie is not sent on cross-site requests, but is sent when the user is navigating to the origin site from an external site.
- **None** means that the browser sends the cookie with cross-site requests. The **Secure** attribute must also be set in this case.

# HTTP Overview - Cookies

**Secure** indicates that the cookie is sent to the server only when the request is made with *https:* scheme (except on *localhost*).

**Partitioned** indicates that the cookie should be stored using partitioned storage (can contrasts user tracking).

# HTTP Overview - Cookies

| | |
|---|---|
| Content-Length: | 209 |
| Content-Type: | text/html; charset=utf-8 |
| Date: | Wed, 05 Feb 2025 21:44:32 GMT |
| Location: | http://flagify-1.challs.cyberchallenge.it/ |
| Server: | nginx/1.18.0 (Ubuntu) |
| Set-Cookie: | session=eyJ1c2VyIjoiYXVzZXJ1c2VyMTEifQ.Z6PbwA.WRXS1aaCyjth__7REGvf7dU88Xs; Domain=flagify-1.challs.cyberchallenge.it; HttpOnly; Path=/; SameSite=Lax |
| Vary: | Cookie |

# HTTP Overview - JWT cookies

**JSON Web Token** is a new standard for managing authentication.

When logged in, the client receive a JSON signed and/or encrypted and encoded in Base64 that holds various informations about the user.


```
eyJhbGciOiJIUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM
0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM
5MDIyfQ.SflKxwRJSMeKKF2QT4
fwpMeJf36POk6yJV_adQssw5c
```


HEADER

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

# Tooling

# Tools

- A browser
- Curl & wget
- Python Requests
- Burp Suite
- Ngrok & Requestbin
- Postman
- Python http.server

# The End