Web Security - Introduction

Stefano Calzavara

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

Greetings!

Stefano Calzavara, PhD

- associate professor at UNIVE and former co-leader of OWASP Italy (2021 - 2023)
- main research interests:
  - **web security**
  - formal methods for security
  - adversarial machine learning
- see my homepage for details

Course Overview

Web Technologies

A primer on key web technologies, such as HTTP, HTML, JavaScript...

Client-Side Web Security

Defending web applications against attacks based on malicious HTML and JavaScript

Server-Side Web Security

Defending web applications against attacks targeting their back-end

Web Protocols

HTTPS, OAuth 2.0, OpenID Connect

Teaching Material

The course is primarily built around:

- Slides: please attend the lectures, take notes, ask questions!
- Technical documentation: somewhat dry, but a necessary evil
- Research papers: they contain way more information than what is required for the final exam, but they may occasionally come in handy

Books

Suggested optional books:

- Stuttard & Pinto - The Web Application Hacker's Handbook (2011)
- McDonald - Grokking Web Application Security (2024)

Exam & Grading

The exam will consist of:

- A written exam including open and closed questions (30 pts)
- Optional assignments, to be solved during the course (3 bonus pts)
- The exam is passed if the overall mark is at least 18

If you are taking the Internet Security course (Web Sec + Network Sec):

- You can take the two exams in any order, even at different times
- You must pass both exams before the next edition of the course
- Each exam grants up to 32 points and the final mark will be the average of the two marks (rounded up)

Advice

Some advice to get the most of this course:

1. Ask questions and interact with the teacher, both during and after the lectures (office time: Friday, 4 PM - 6 PM, by appointment)
2. You have a tutor to interact with: Simone Bozzolan (put me in Cc)
3. Carefully attend the lectures and take notes (important!)
4. Do the assignments: they are optional, yet invaluable for learning
5. Be proactive: try out things by yourself, take a look at the extra material linked in the slides and feel free to search for more
6. Bring your own laptop: several classes with hacking labs are planned!

Ethics

In this course we are discussing vulnerabilities that might affect existing software and services...

- Do not look for vulnerabilities in services that you do not own!
- You can look for vulnerabilities in local installations of free software, however, if you find one, it should be responsibly disclosed
- If you are not familiar with responsible disclosure, reach back to me, but really: be careful about what you are doing
- Do you want to hack? Tons of labs, challenges and CtFs around

Regarding assignments: you are encouraged to exchange ideas with your colleagues, but do not cheat and do not copy-paste solutions!

The Web Platform

Stefano Calzavara

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

Web Applications

A web application is a client-server distributed application (normally) operated via a web browser:

- e-commerce sites: Alibaba, Amazon, Ebay
- mail services: Gmail, Outlook, Yahoo!
- social networks: Facebook, Instagram, Twitter

Fantastic tools which have become more and more popular over the years, yet extremely hard to protect!

Web Security: Challenges

Sheer Value of Web Assets

The more the Web integrates with our everyday life, the more web assets become primary attack targets!

Open Nature of the Web

Web applications are often publicly accessible by design, so anyone can act as the attacker!

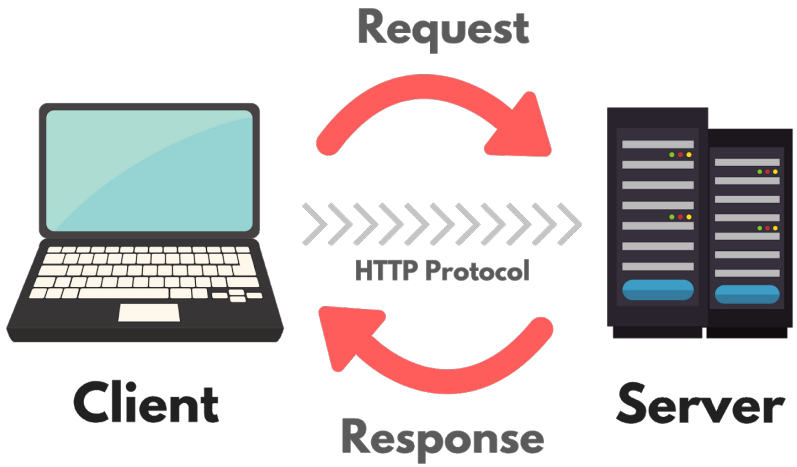
Web Complexity

Many different technologies involved, even for the same task, and often with a complicated semantics!

Hyper Text Transfer Protocol (HTTP)

The HTTP protocol is the workhorse protocol of the Web:

- simple request-response protocol with a client-server model
- plaintext: no confidentiality, integrity or authenticity guarantee
- stateless: each HTTP request is handled as an independent event, i.e., the server has no memory of previous requests



Uniform Resource Locator (URL)

Web resources are identified by a URL, defined by the following syntax:

```
scheme://[user@]host[:port]path[?query][#fragment]
```

where:

- `scheme` identifies a protocol (normally: HTTP or HTTPS)
- `user` contains authentication credentials, e.g., `alice:secret`
- `host` identifies the remote server (via IP address or domain name)
- `port` is normally 80 for HTTP and 443 for HTTPS
- `path` identifies the resource on the server (slash-separated strings)
- `query` binds parameters to values (`p1=v1&p2=v2...`)
- `fragment` normally identifies a part of the HTML page (anchor) and is not sent to the server

URL Encoding

URL encoding is a technique to encode arbitrary data into the ASCII alphabet adopted in the syntax of URLs:

- also required when using reserved characters like / and ? as literals, as opposed to their special meaning
- URLs differing only by whether some unreserved character (like A) is URL-encoded or not are considered equivalent
- the notion of "reserved" is context-dependent, e.g., / is reserved in the path, but not in the query string (be careful!)

Example

"This contains /" is encoded as "This%20contains%20%2F"

URL Encoding

Are these URLs valid? Are they pairwise equivalent?

URL1	URL2
`http://a.com/abc.php`	`http://a.com/ab%63.php`
`http://a.com/abc.php`	`http://a.com%2Fabc.php`
`http://a.com/?b#c`	`http://a.com/?b%23c`

Notes:

- use the URL class of JavaScript to see how URLs are parsed
- use the `encodeURIComponent` and `decodeURIComponent` functions to perform URL encoding and decoding respectively
- parsing and then comparing the decoded URL components is a good way to understand whether two URLs are equivalent or not

URL Encoding

```
function areURLsEquivalent(url1, url2) {
    try {
       const u1 = new URL(url1);
       const u2 = new URL(url2);

       return (
           decodeURIComponent(u1.origin) === decodeURIComponent(u2.origin) &&
           decodeURIComponent(u1.pathname) === decodeURIComponent(u2.pathname) &&
           decodeURIComponent(u1.search) === decodeURIComponent(u2.search) &&
           decodeURIComponent(u1.hash) === decodeURIComponent(u2.hash)
        );
    } catch (e) {
       return false; // If one of the URLs is invalid
    }
}
```

Domain Names

On the Web, the server is typically identified by a string known as fully qualified domain name (FQDN).

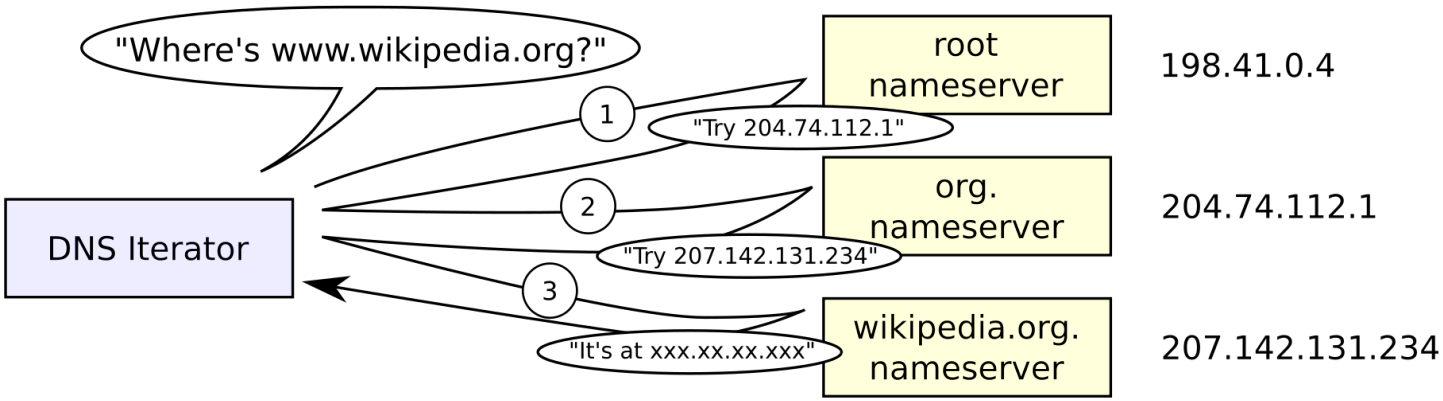
Terminology

The string `www.wikipedia.org` is a FQDN and:

- `wikipedia.org` is a domain name (or just domain for short)
- `www` is a subdomain of the domain `wikipedia.org`
- `org` is a top-level domain, like `com` or `it`

The Domain Name System (DNS) protocol is used to resolve a domain name into an associated IP address, which identifies a single host.

Domain Name System (DNS)



Notes:

- A domain with at least one IP address is also called hostname
- Many-to-many mapping between domain names and IP addresses
- The output of DNS resolution is stored in a local cache, up to a TTL

Registrable Domains

Domains need to be registered by paying a fee to a registrar, such as Aruba or GoDaddy. Registrable domains are defined as a public suffix[1] plus one additional label:

- registrable domains are also called sites in browser security
- the owner of a site has control of the DNS resolution of the site and all its subdomains: different sub-domains can resolve to different IP addresses chosen by the site's owner

Example

- `uk` is a top-level domain and `co.uk` is a public suffix
- `bbc.co.uk` is a registrable domain or site
- `foo.bbc.co.uk` and `bar.bbc.co.uk` are subdomains of `bbc.co.uk`

[1] `https://publicsuffix.org`

HTTP Requests

HTTP requests are structured as follows:

1. a request line, including method, resource and protocol version
2. a list of request headers, including at least the `Host` header
3. an empty line `<CR><LF>`, acting as separator
4. an optional request body

Example

```
POST /cart/add.php HTTP/1.1<CR><LF>
Host: www.amazon.com<CR><LF>
<CR><LF>
item=56214&quantity=1
```

HTTP Methods

The most common methods available in HTTP:

- GET: retrieves information from the server
- HEAD: like GET, but does not retrieve the response body
- POST: sends data to the server for processing
- PUT: uploads data (file) to the server
- DELETE: removes data (file) from the server
- OPTIONS: asks for the list of supported methods

HTTP Methods

Method	Req. body	Resp. body	Safe	Idempotent
GET	optional	yes	yes	yes
HEAD	optional	no	yes	yes
POST	yes	yes	no	no
PUT	yes	yes	no	yes
DELETE	optional	yes	no	yes
OPTIONS	optional	yes	yes	yes

- Safe = the server's state does not change
- Idempotent = processing multiple times is equivalent to just once
- First web security insight: do not trust this table!

HTTP Responses

HTTP responses are structured as follows:

1. a status line, including status code and reason message
2. a list of response headers
3. an empty line <CR><LF>, acting as separator
4. an optional response body

Example

```
HTTP/1.1 200 OK<CR><LF>
Content-Type: text/html; charset=UTF-8<CR><LF>
<CR><LF>
<html><body>Done!</body></html>
```

HTTP Status Codes

Code	Category	Example
2XX	Success	200 OK
3XX	Redirection	301 Moved Permanently
4XX	Client error	401 Unauthorized
5XX	Server error	503 Service Unavailable

- Again: this is just a convention, that the server can ignore!
- The only status codes with a special meaning are redirects, whose target is set in the `Location` header of the response: the browser will immediately send a new request to the target therein.

Response Body

The response body is just text. Yet, some text has special meaning...

Hyper Text Markup Language (HTML)

Markup language used to define the structure of a web page:

- parsed as a DOM tree by the browser before rendering
- formatted for presentation by using Cascading Style Sheets (CSS)

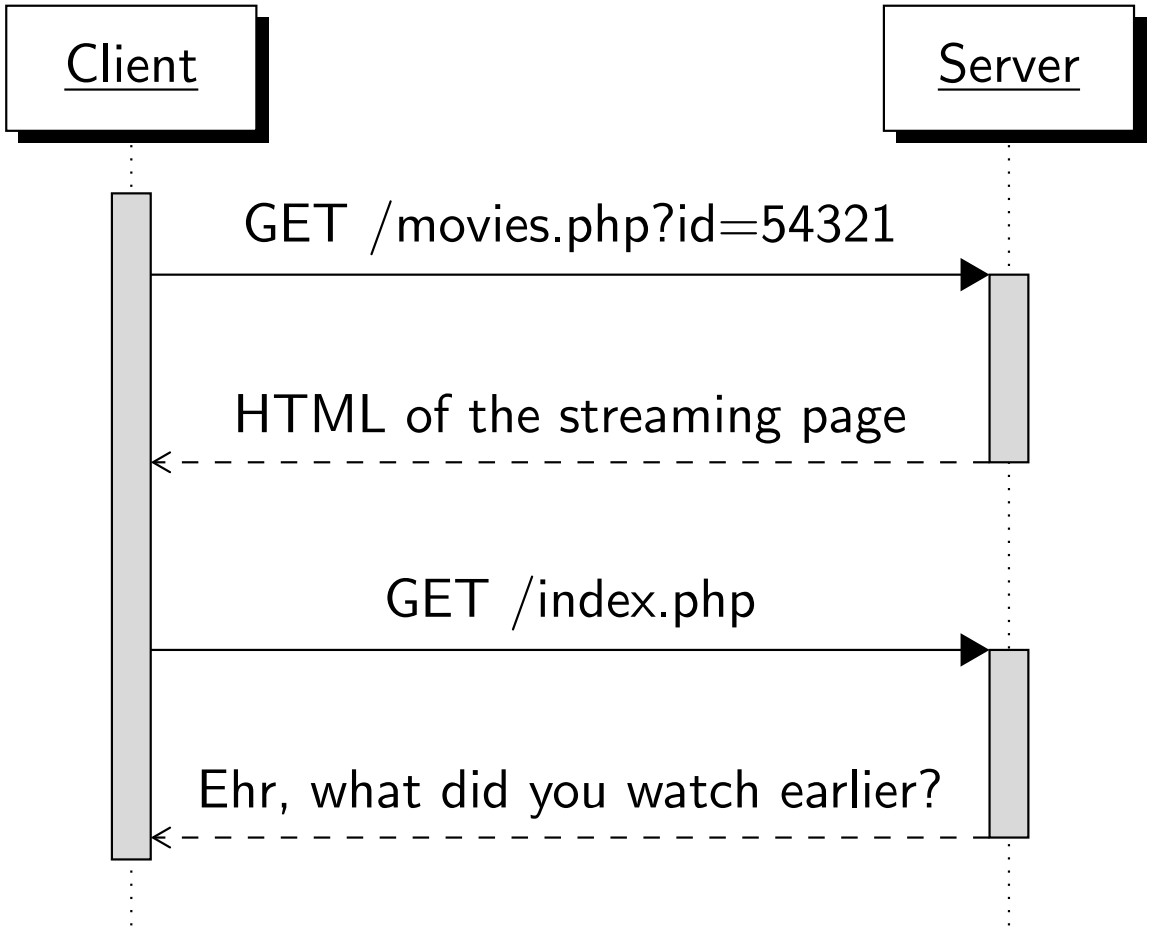
JavaScript

Client-side scripting language:

- full-fledged programming language, used basically on every website
- enriched with powerful APIs, which enable DOM manipulations

HTTP State Management

Recall that HTTP is a stateless protocol, i.e., the server has no memory!



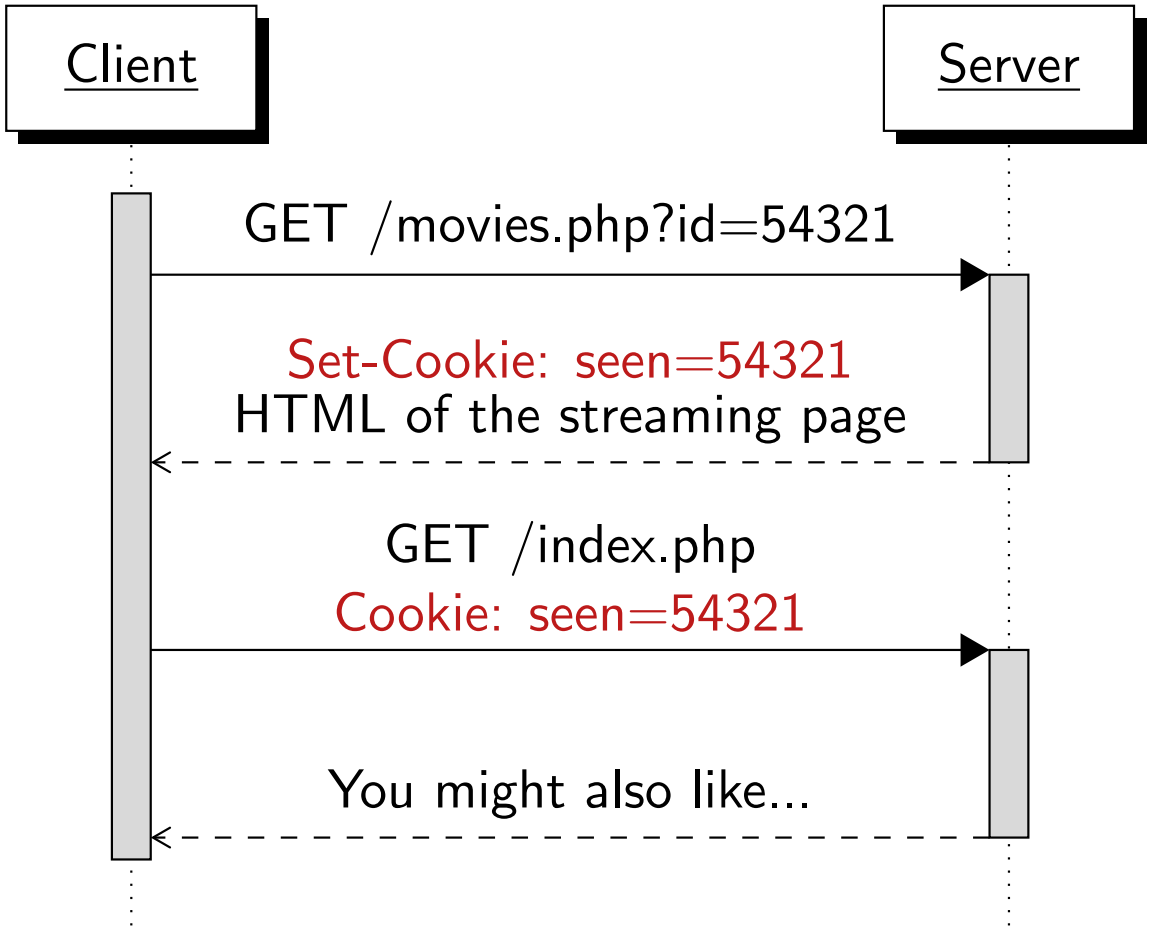
HTTP State Management

State information can be stored at the client by means of cookies:

- a cookie is just a small piece of data of the form `key=value`, e.g., `lang=en` can be used to store language preferences
- first set by the server into the client via an HTTP response, using the `Set-Cookie` header
- then automatically sent by the client to the server inside HTTP requests, using the `Cookie` header
- the server can inspect the cookie and process it before crafting the next HTTP response, e.g., to provide a personalized user experience

Cookies are opaque: the server can set any information inside a cookie in whatever format!

HTTP State Management



Cookie Scope

Cookies are normally attached to all the HTTP requests towards the FQDN of the server which originally set them.

- The scope of a cookie can be restricted to specific paths using the `Path` cookie attribute.
- Web applications hosted on the same site can share cookies, using the `Domain` cookie attribute. This allows sharing state information across multiple subdomains owned by the same organization.

Example

A web application at `accounts.example.com` can set a cookie with the `Domain` attribute set to `.example.com`, which is shared with all its sibling domains like `mail.example.com` (and their children).

Cookie Scope

Cookie scoping is regulated by public suffixes: it is restricted to the same-site position and one cannot set cookies having an overly large scope, e.g., with their `Domain` attribute set to `.com`

Setter	**Domain**	**Destination**	**Outcome**
www.foo.com	-	www.foo.com	Sent
www.foo.com	-	sub.foo.com	Not sent
www.foo.com	-	foo.com	Not sent
www.foo.com	.foo.com	www.foo.com	Sent
www.foo.com	.foo.com	sub.foo.com	Sent
www.foo.com	.foo.com	foo.com	Sent
www.foo.com	.com	www.foo.com	Forbidden

HTTP Secure (HTTPS)

HTTPS is the secure counterpart of HTTP:

- encrypted variant of HTTP based on the TLS protocol
- ensures confidentiality and integrity of the HTTP messages
- provides authentication of the server through signed certificates

Since 2017 the amount of HTTPS traffic surpassed the amount of HTTP traffic, based on public data from Mozilla.[2]

Alert!

HTTPS is necessary, but not sufficient, for web application security!

We will discuss HTTPS in detail later in the course.

[2] `www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer`

Attacking the Web

The most common question in security: what can go wrong?

Web Attacker

- owner of malicious website
- attacks via HTML & JavaScript
- we assume the victim accesses the attacker's website
- baseline attacker model: always take it into account!

Network Attacker

- owner of the network
- full control of HTTP traffic
- we assume the victim accesses some HTTP site at some point
- still a realistic attacker and very relevant for major websites

Other attackers have been studied in the literature, but these are the most significant in the majority of settings.