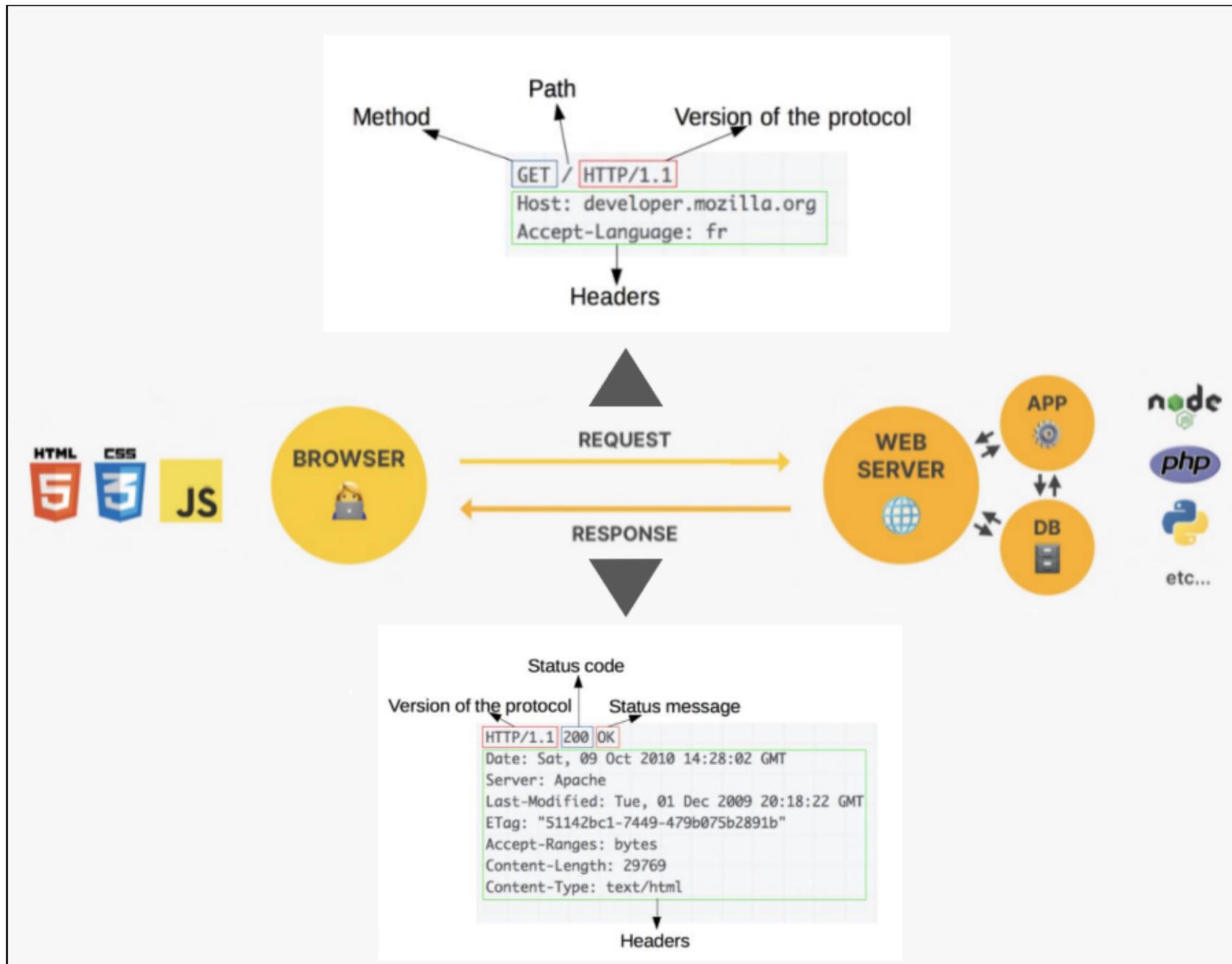


# 6. Segurança Web

## 6.1. Protocolo HTTP

- HTTP: Hyper Text Transfer Protocol
- Protocolo que usa logical links / hyperlinks, ou seja, que permite num determinado documento referenciar outras páginas web, documentos ou elementos específicos. O utilizador pode seguir essa hyperlink, clicando nesse link.
- Um recurso é identificados por uma localização uniforme (URL):
  - esquema ou protocolo (http, https, etc.)
  - domínio (aaa.bbb.cc), potencialmente com uma porta específica
  - caminho para o recurso (path)
  - informação extra (e.g., informação de query ou identificador de fragmento)
  - Exemplo:  
[https://sigarra.up.pt/fcup/en/cur\\_geral.cur\\_view?pv\\_curso\\_id=6041&pv\\_ano\\_lectivo=2021](https://sigarra.up.pt/fcup/en/cur_geral.cur_view?pv_curso_id=6041&pv_ano_lectivo=2021)
- Comunicação efetuada através de pedidos como:
  - GET: obter recurso numa URL específica (não deve ser usado para alterar o estado do servidor pois tem sempre side-effects)
  - POST: criar um novo recurso numa URL específica (utilizado para quase tudo o que originalmente era previsto para PUT, PATCH e DELETE)
  - PUT: substituir representação de recurso existente com outro conteúdo
  - PATCH: alterar parte de um recurso
  - DELETE: apagar recurso numa URL específica.



## 6.2. Cookies

- Forma de as aplicações do servidor reconhecerem pedidos relacionados.
- Uma cookie é um bocado de informação que uma aplicação web (no servidor) pode pedir ao browser para armazenar.
- Útil para gestão de sessões, personalização, rastreamento/profiling.
- Passos:
  1. O servidor pede ao browser para armazenar informação.
  2. Essa informação é guardada num ficheiro.
  3. Sempre que o browser volta a pedir um recurso ao mesmo servidor, devolve esse ficheiro (a cookie).
- Se definirmos uma cookie num determinado domínio, todos os sub-domínios têm acesso a essa cookie.

- Só posso definir cookies para a minha origem ou para domínios que estão acima hierarquicamente da minha origem. Sempre que algum recurso é pedido ao servidor e que faz match com essa origem, a cookie é enviada.
- Exemplo: uso de cookies para autenticação
  - Um servidor quer autenticar um cliente com username e password. Depois quer lembrar-se que esse utilizador está autenticado.



1. Utilizador autentica-se com username e password.
2. O servidor verifica se as credenciais estão corretas e guarda a sessão em memória.
3. Servidor envia a resposta e pede ao cliente para criar uma cookie.
4. A cookie é criada do lado do cliente, pelo browser.
5. Em futuros pedidos que o cliente faça a esse servidor, o browser envia a cookie.
6. O servidor vê pela cookie que recebeu se é o mesmo cliente que se autenticou no passo 1.
7. Envia a resposta ao pedido (recordando a sessão e eliminando a necessidade de o utilizador se autenticar sempre).

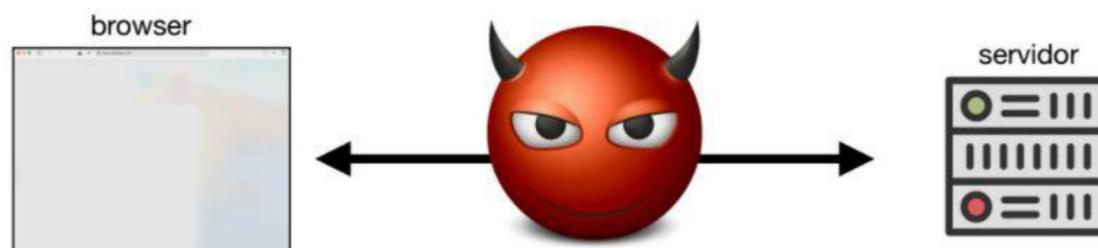
### 6.3. Modelo de execução

- O modelo de funcionamento de execução web é um modelo complexo, pois estamos a considerar uma aplicação - o browser - onde correm várias sub-aplicações (iFrames) que vêm de diversas origens e o browser está de alguma forma a tentar garantir que essas aplicações não interferem umas com as outras.
- A segurança web está maioritariamente na forma como o browser gere contextos de confiança.

- Noção de **origem**: o protocolo que está a ser utilizado (por exemplo, HTTP) e o domínio.
- JavaScript consegue aceder a todos os recursos que vêm da mesma origem, incluindo a própria origem do JavaScript. No entanto, o javascript também consegue executar código de outra origem (outra página onde fui buscar esse código) se for executado no meu contexto, ou seja, em meu nome.
- **Computações do lado do servidor:**
  - bases de dados, gestão de sessões, personalização, etc.
- **Computações do lado do cliente:**
  - Os browsers são pequenos sistemas de virtualização, onde cada janela:
    - processa respostas HTML
    - executa JavaScript se necessário
    - efetua pedidos para sub-recursos (imagens, CSS, JavaScript, etc.)
    - responde a eventos do utilizador ou eventos definidos pelo próprio site
- Uma janela do browser pode ter várias tabs com conteúdos de diferentes origens.
- Cada tab é uma **frame** e possui várias sub-frames ou **iFrames**.
- **iFrames**: elementos HTML que permite embeber uma página web dentro de outra página web.
- **Vantagens de frames e/ou iFrames:**
  - Delegar uma área do ecrã para outra origem, ou seja, numa secção do website mostrar ao utilizador conteúdo de outra origem, por exemplo, anúncios.
  - O browser impõe as regras de isolamento entre as frames. Pode-se aproveitar essa feature dos browsers para proteger os dados do utilizador.
  - Devido ao isolamento entre as iFrames e/ou frames, a página mãe não colapsa se uma sub-frame (iFrame) falhar.
- O código JavaScript pode ler e alterar o estado de uma página com o **DOM**: interface orientada a objetos, que representa toda a página de forma hierárquica e que permite alterar, inspecionar e adicionar elementos dessa página (imagens, cookies, etc).

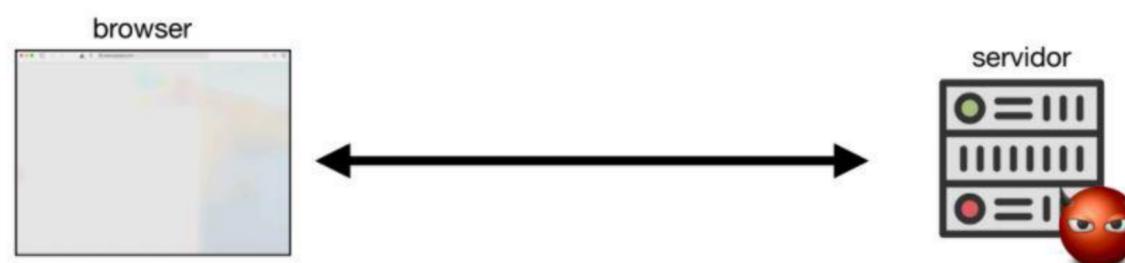
## 6.4. Modelos de ataque

- **Atacante externo/rede:** Adversário que controla apenas o meio de comunicação.

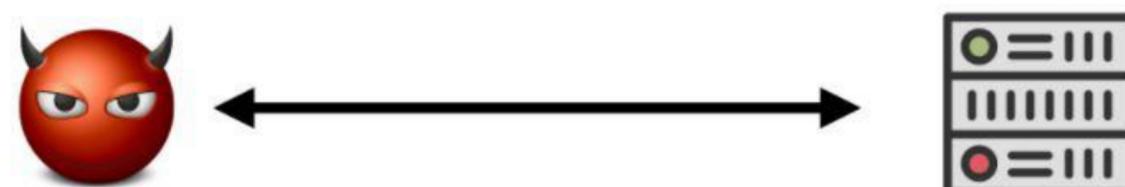


- **Atacante interno/web:** Adversário que controla parte da aplicação web. Tem várias variantes:

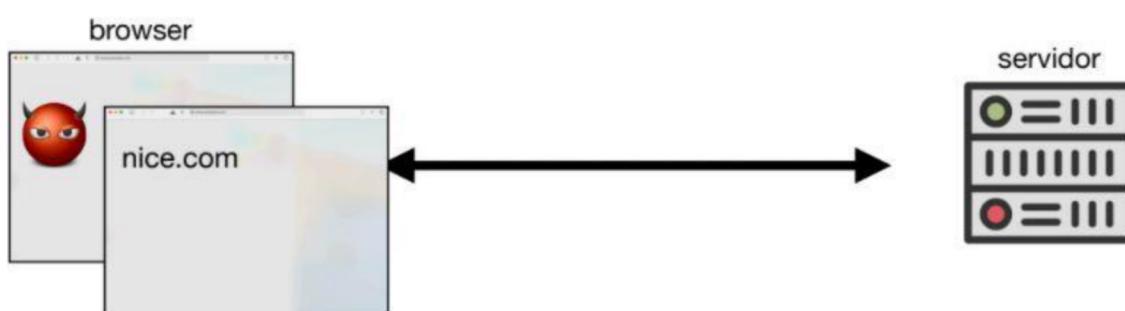
- Adversário que controla servidor.



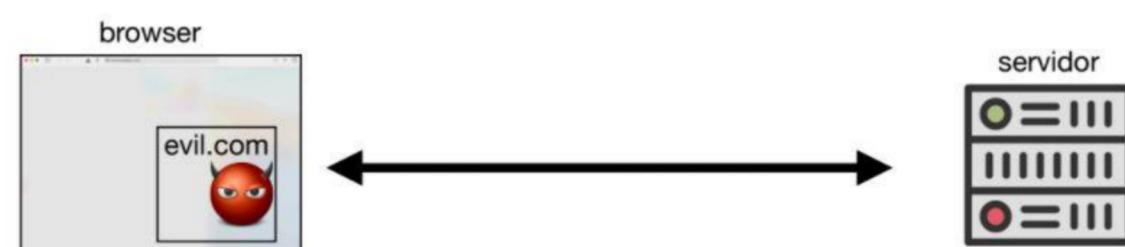
- Atacante que controla cliente.



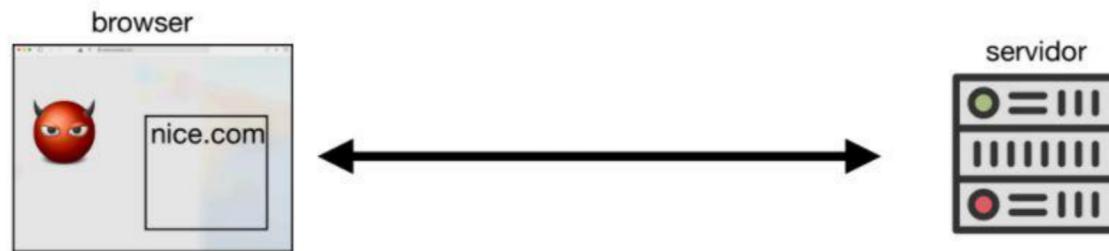
- Adversário que controla uma página no cliente.



- Adversário que controla um objeto embedido numa página.



- Adversário que controla uma página e que pode interferir com o objeto embeddo nessa página.



## 6.5. Modelo de segurança

### 6.5.1. Same Origin Policy (SOP)

- Política de isolamento imposta pelo browser que providencia:
  - confidencialidade: dados de uma origem não podem ser acedidos por código de origem diferente
  - integridade: dados de uma origem não podem ser alterados por código com uma origem diferente
- **SOP no DOM:**
  - Cada frame tem uma origem (esquema, nome de domínio, porta)
  - Código numa frame só pode aceder a dados com a mesma origem
- **SOP para mensagens:**
  - Frames podem comunicar entre si!
- **SOP para cookies:**
  - No contexto das cookies, a definição de origem tem apenas em conta o domínio e o path (o esquema é opcional).
  - A origem que o browser associa a uma cookie é aquilo que a cookie declara quando é criada.
  - Uma página pode definir uma cookie para:
    - o seu domínio
    - domínio hierarquicamente superiores (exceto sufixos públicos como `.com`, `.org`, etc)

- **Quando é que o browser envia as cookies?**
  - As cookies apenas são enviadas pelo browser para servidores com a mesma origem que as criou.
  - Se SameSite = None, as cookies são enviadas:
    - se o domínio da cookie for um sufixo do domínio da URL
    - se a path da cookie for um prefixo da path da URL
    - Apenas comparo o URL do pedido com o domínio e path da cookie.

		Do we send the cookie?		
Request to URL		Set-Cookie: ...; Domain=login.site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/my/home;
checkout.site.com		No	Yes	No
login.site.com		Yes	Yes	No
login.site.com/my/home		Yes	Yes	Yes
site.com/my		No	Yes	No

- Se SameSite = Strict, as cookies são enviadas:
  - apenas quando o pedido tem a mesma origem que a top-level frame (página principal)
  - Apenas comparo o domínio do URL do pedido com o domínio da frame que está a pedir.
- Se SameSite = Lax, as cookies são enviadas:
  - quando o utilizador está a navegar no site de origem (por exemplo, seguindo/clicando num link). Ou seja, todos os links que estiverem presentes num dado site terão acesso às cookies, sem restrições ou comparações de origens.
  - Permite ataques como CSRF.
- Com **secure cookies**, as cookies são enviadas:
  - apenas por HTTPS (para evitar que as alguém as inspecione na rede)