Autenticação em aplicações Web

- Revisão do protocolo HTTP
- *Cookies*
- Autenticadores em contexto web

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

1

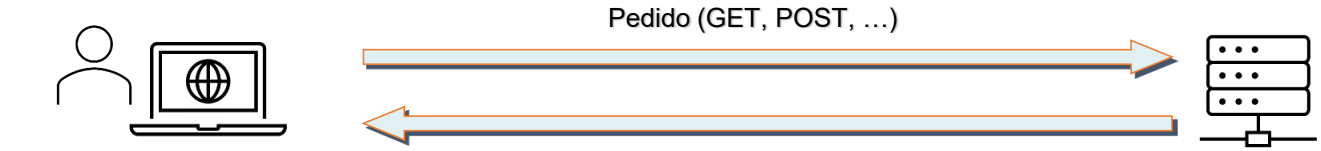
Revisões HTTP

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

2

Hypertext Transfer Protocol (HTTP)

- Objectivo original: Transferir documentos em hiper-texto (HTML)
- Evolução do protocolo coordenada pelo W3C
  - RFC 1945 - Versão 1.0
  - RFC 2616 - Versão 1.1
  - RFC 7540 – Versão 2 (binário e múltiplos pedidos na mesma ligação)
- Principais caracteríristica: sem estado (*stateless*) mas com possibilidade de manter sessões
- Servidor recebe pedidos TCP no porto 80 (por omissão)



User Agent
(por exemplo, Web Browser)

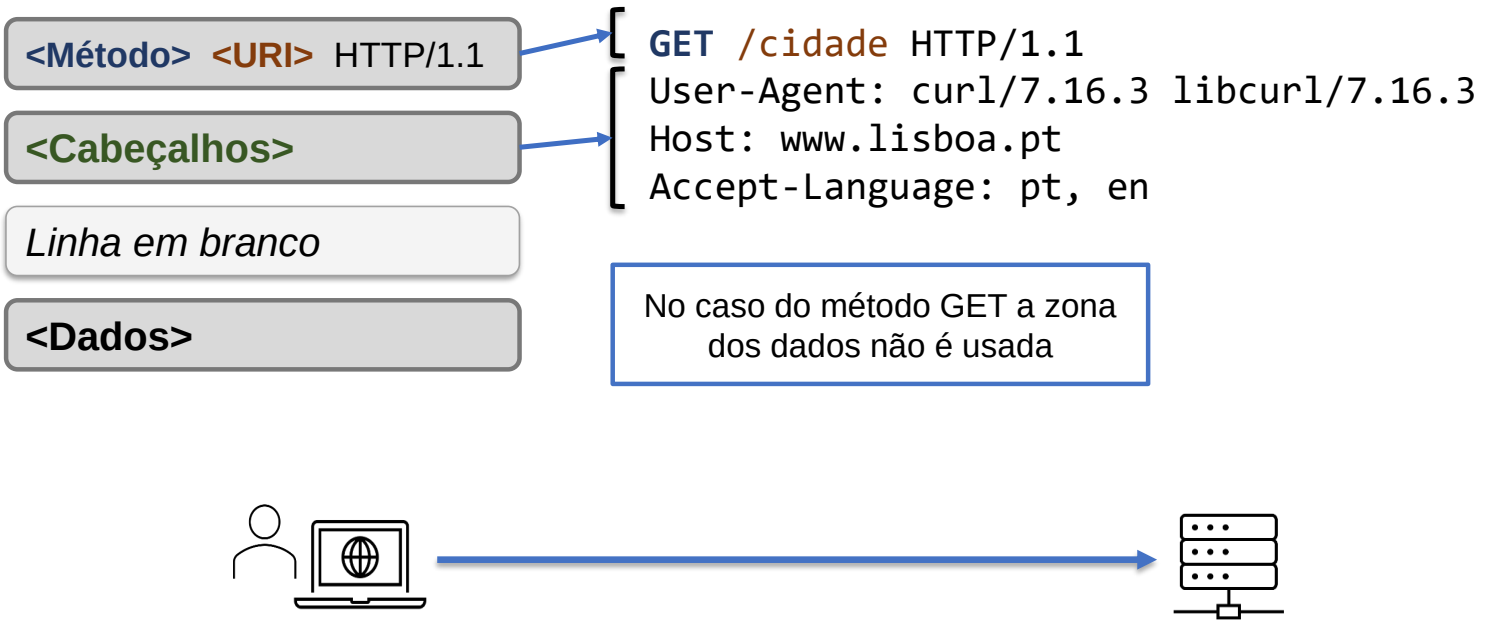
Pedidos HTTP

Tipo de pedidos *(Request Methods)*	Descrição
GET	Pedido para obter a representação de um recurso (método mais comum).
HEAD	Pedido idêntico ao GET mas o corpo da resposta não é enviado (apenas os *headers*).
POST	Envio de dados para um recurso. Os dados vão no corpo do pedido.

- Outros tipos de pedidos: PUT, DELETE, TRACE, OPTIONS

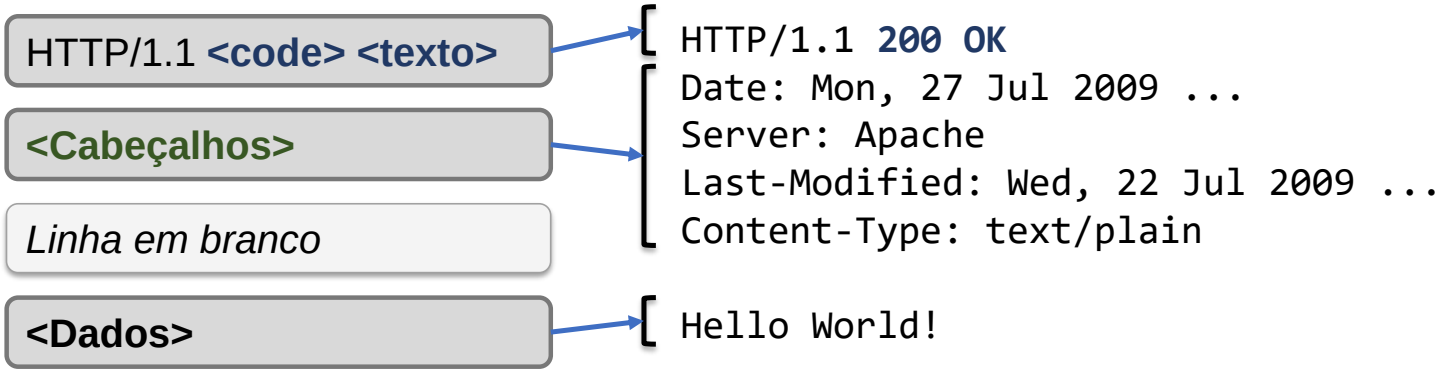
Formato de um pedido HTTP

Exemplo com GET



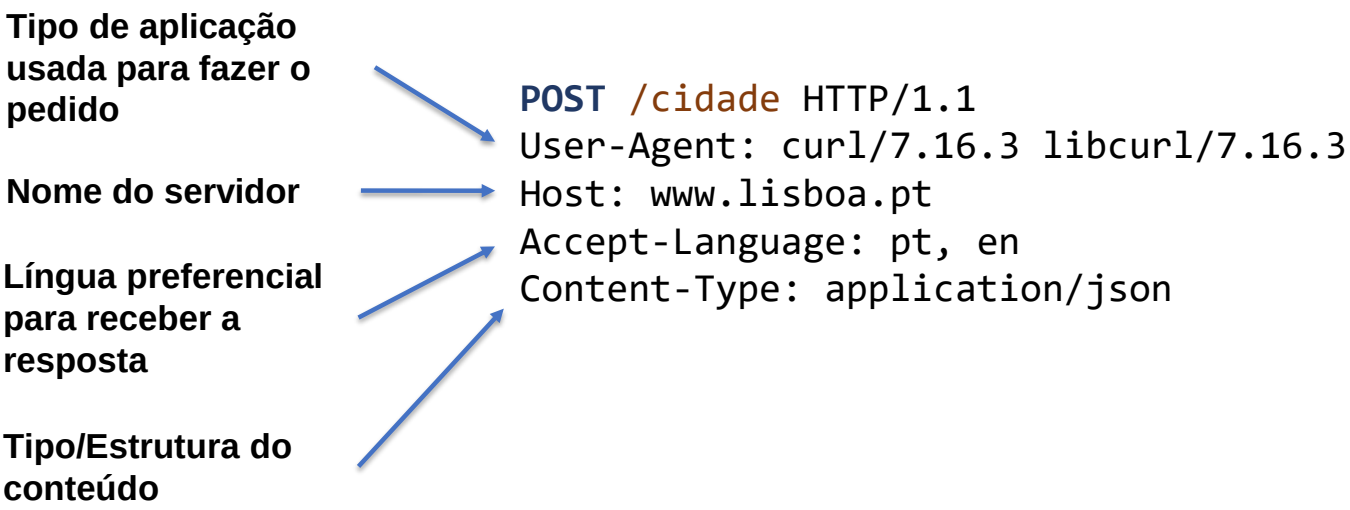
Formato de uma resposta HTTP

Exemplo de resposta



Cabeçalhos (alguns exemplos)

- Nos pedidos e respostas está previsto o uso de cabeçalhos para transportar informação adicional sobre o pedido/resposta



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

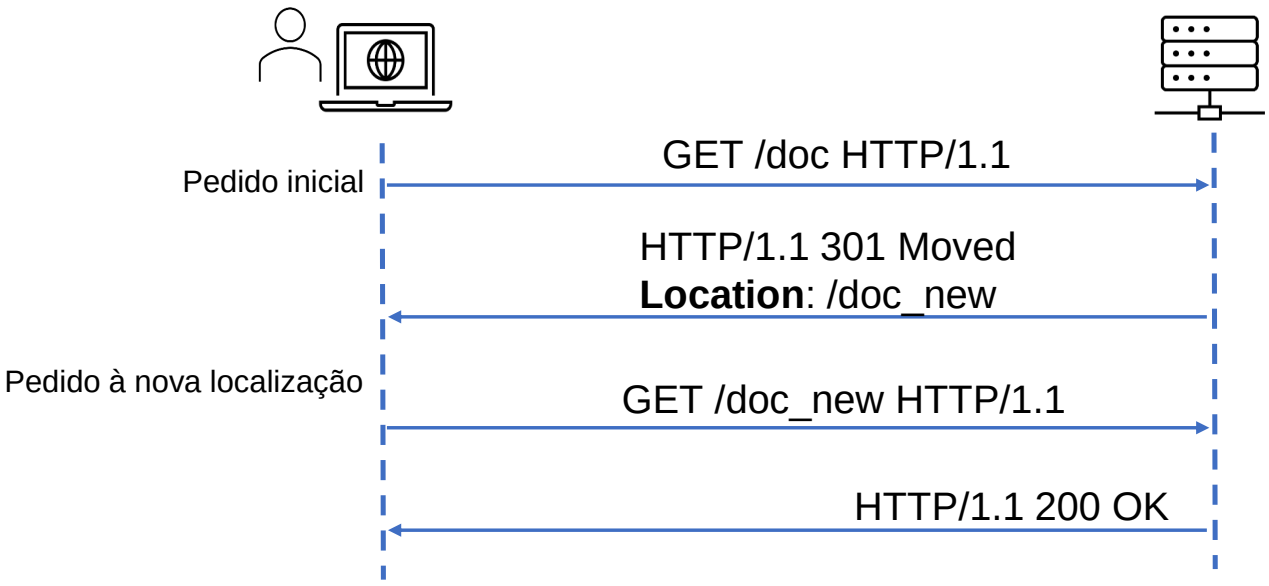
Códigos de resposta (alguns exemplos)

- 1xx – Informação
- 2xx – Sucesso
  - **200 OK**
  - 204 No Content
- 3xx – Redirecionamento
  - 302 Found
- 4xx – Erro de cliente
  - 400 Bad Request
- 5xx – Erro de servidor
  - 500 Internal Server Error

https://datatracker.ietf.org/doc/html/rfc7231#section-6.1

Resposta `Redirect`

- A redireção é desencadeada pelo servidor enviando um código de resposta `3xx`, *header* de `Location` referindo o URL que o `user-agent` deve pedir



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

Manutenção de estado em aplicações Web

- **O protocolo HTTP é *stateless***
  - Os pedidos são independentes e sem relação (mesmo pedidos consecutivos do mesmo cliente na mesma ligação TCP)

- Como manter uma **conversação** entre o cliente e o servidor em HTTP?
  - Utilizam-se HTTP Cookies

RFC 2965: *HTTP State Management Mechanism*
RFC 2964: *Use of HTTP State Management*

10

Cookies HTTP

- O que são Cookies?
  - Mecanismo que fornece a aplicações HTTP servidoras suporte para guardar e obter informações sobre o cliente
  - Manutenção de informação de estado sobre o cliente

- Que informação contém um Cookie?
  - Informação sobre o estado do cliente na forma de par `nome=valor`
  - 'Range' de URLs para o qual o estado é válido
  - Data de validade (para o caso de Cookies persistentes)

- Utilizados através dos headers HTTP

Utilizações habituais de Cookies

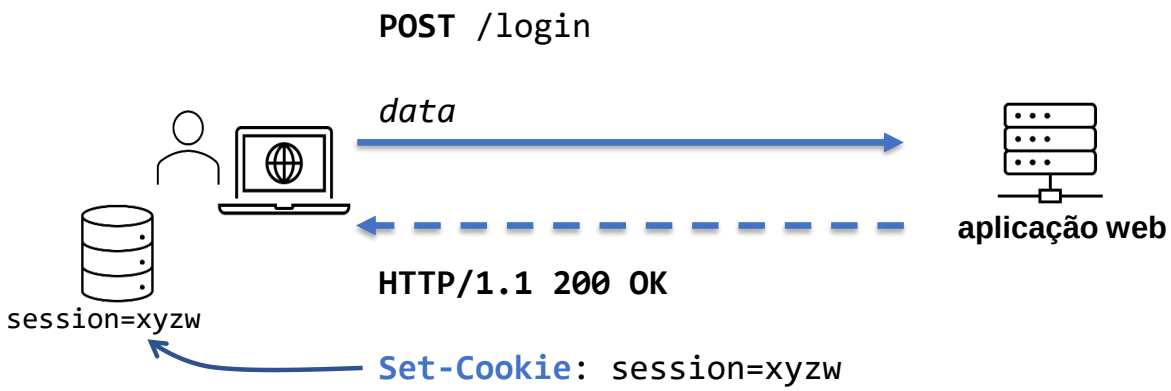
- **Podem ser** utilizados para:
  - Criar sessões (conversação)
  - Evitar *login* (*login* automático)
  - Deixar registo de navegação (incluindo *third-party* cookies)
  - Deixar registo de preferências do cliente

- **Não podem** (não é possível) ser utilizados para:
  - Aceder ao disco rígido
  - Enviar vírus para o cliente

12

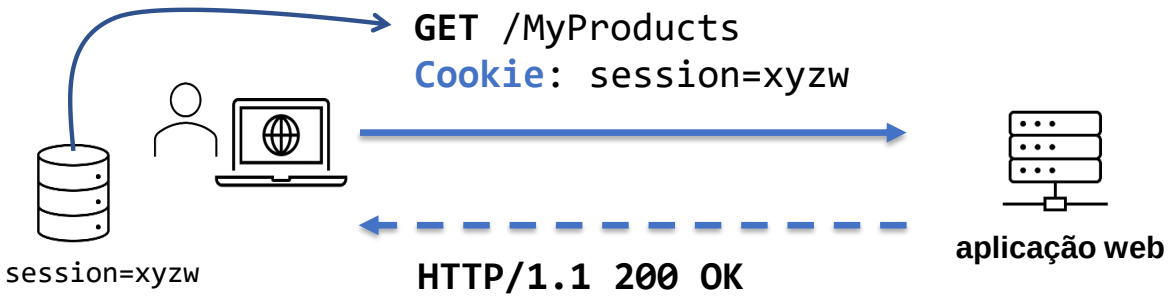
Cookies

- No protocolo HTTP cada pedido é independente do anterior
- No entanto, uma das formas da aplicação web manter um contexto com o utilizador é através de cookies (usando cabeçalhos de resposta e pedido)



Cookies

- Cada vez que é feito um pedido a uma aplicação, o *browser* envia automaticamente os cookies que tiver para essa aplicação
  - O cookie fica associado a um domínio (ex: www.mysite.com) e um caminho dentro desse domínio (ex: / ou /some/path)



14

Headers HTTP referentes aos Cookies

Sintaxe do header 'Cookie' no pedido HTTP

```
Cookie: {<NAME>=<VALUE>;}+
```

Sintaxe do header 'Set-Cookie' na resposta HTTP

```
Set-Cookie: {<NAME>=<VALUE>;}+  [expires=<DATE>;]
[path=<PATH>;] [domain=<DOMAIN_NAME>;] [secure] [httpOnly]
```

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

Algumas notas sobre Cookies

- Uma resposta HTTP pode conter múltiplos cabeçalhos `Set-Cookie`

- O campo expires indica ao cliente quando o Cookie "deve" ser removido
  - No entanto o cliente não é obrigado a removê-lo
  - O cliente pode remover o Cookie antes deste expirar se o número de Cookies exceder os limites internos do cliente

- Para uma aplicação servidora **apagar um Cookie** no cliente, deverá enviar na resposta um Cookie com o mesmo nome e uma data de expiração passada

16

Autenticadores em contexto Web

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

17

Autenticação “basic”em HTTP

- O protocolo HTTP não possui estado
  - A autenticação tem de ser realizada em todos os pedidos
- Dois esquemas de autenticação definidos no RFC 2617 – *basic* e *digest authentication*
- Fluxo de autenticação – *basic authentication*
  - Cliente acede a recurso protegido
  - Servidor responde com 401 e response header WWW-Authenticate
  - Cliente acede ao recurso usando o header Authorization com a codificação base 64 do utilizador e password

Na maior parte dos casos é usado apenas esta interação, com valores no `Authorization` definidos pela aplicação web

C->S	`GET /docu2.html HTTP/1.1`
C<-S	`HTTP/1.1 401 Unauthorized` **`WWW-Authenticate`**`: Basic realm="testrealm@host.com"`
C->S	`GET /docu2.html HTTP/1.1` **`Authorization`**`: Basic <base 64 of userid “:” password>`

18

Fases de autenticação

- O esquema de autenticação do HTTP não resolve a manutenção do estado de autenticação
  - O browser tem de guardar a *password* ou pedi-la em cada pedido.
  - O servidor tem de validar a *password* em cada pedido
- Alternativa: usar **autenticadores**

- Duas fases na autenticação
- Fase 1
  - Apresentação das credenciais pelo utilizador (ex.:“username+password”)
  - Obtenção dum autenticador gerado pelo servidor
- Fase 2
  - Apresentação do autenticador automaticamente pelo “user-agent”

19

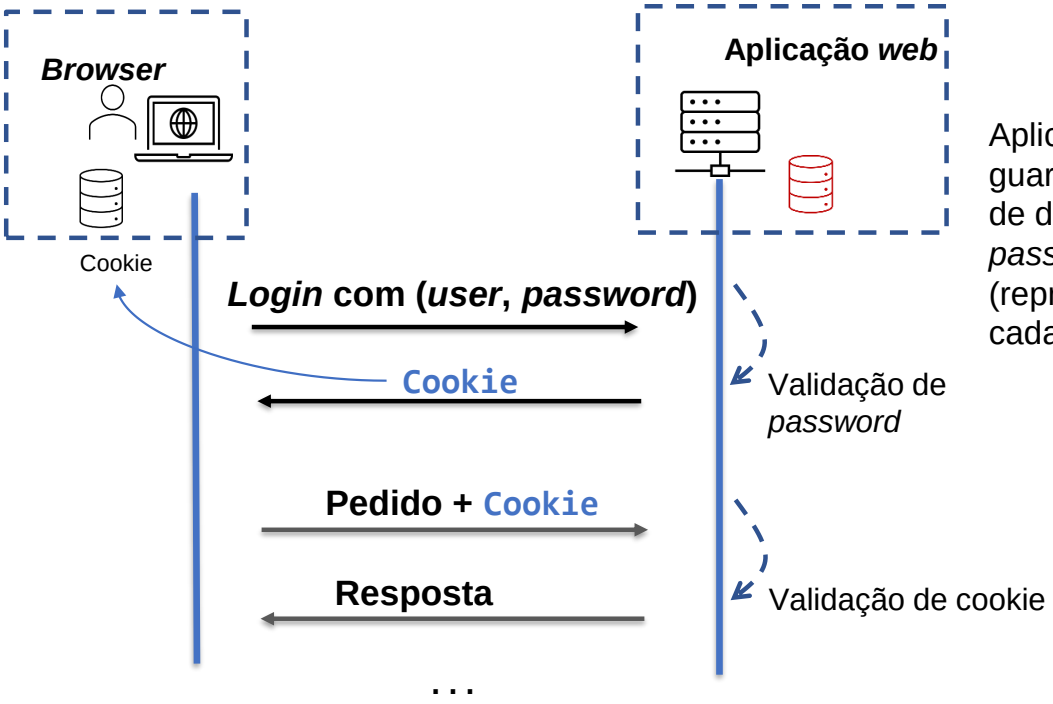
Objectivos do atacante

- Falsificação existencial
  - Obter um autenticador válido
- Falsificação seletiva
  - Dado um utilizador, obter um autenticador válido para esse utilizador
- Obtenção da chave usada na criação de autenticadores

20

Exemplos de Autenticador: *cookie*

Utilizador passa por um processo de registo onde escolhe nome de utilizador e *password*



Aplicação web guarda numa base de dados a *password* (representante) de cada utilizador

21

Autenticadores com *cookies*: implementação

- Identificador de sessão
  - Informação sobre a sessão presente no servidor
  - "Cookie" contém o identificador para aceder a essa informação
  - Deve ser computacionalmente infazível criar um identificador válido
    - Geração criptográfica de números aleatórios

- Message Authentication Code
  - Informação sobre a sessão presente no cookie
  - Cookie protegido por um MAC
  - Se a confidencialidade for requisito, cifrar o conteúdo do cookie

22

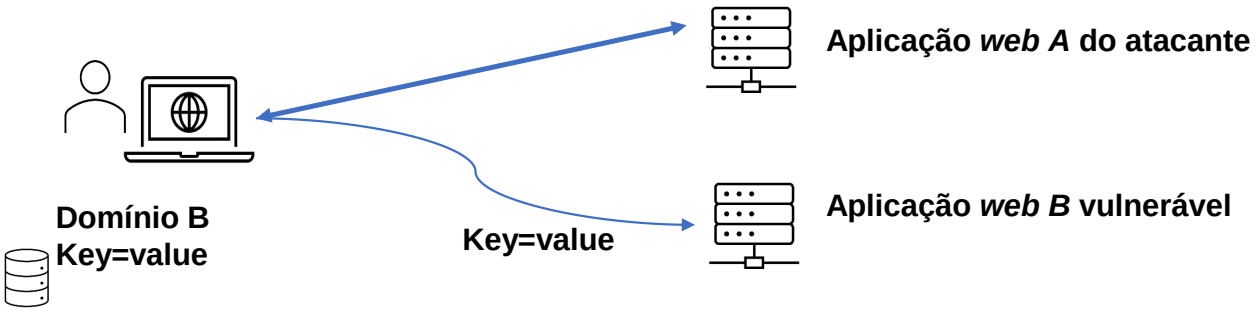
Autenticadores com *cookies*: implementação

- Mecanismo de validade temporal próprio
  - Não se deve utilizar o mecanismo dos cookies
  - Limite da validade temporal
    - Presente na informação de sessão
    - Presente no cookie (protegido por esquema **MAC**)
- *Logout*/Revogação
  - Invalidar a sessão
  - Colocar o *cookie* numa lista de revogação (até à expiração da validade)
- Protecção dos cookies
  - Protecção do transporte através do uso de SSL (Uso da opção *flag* Secure)
  - Proteção no cliente – *flag* HttpOnly

23

Cookies e segurança

- A aplicação web valida o acesso com base na autenticidade dos cookies
- A falsificação de um pedido que mude o estado da aplicação pode ser feita usando pedidos *cross-site*
  - Problema conhecido como *Cross-Site Request Forgery*

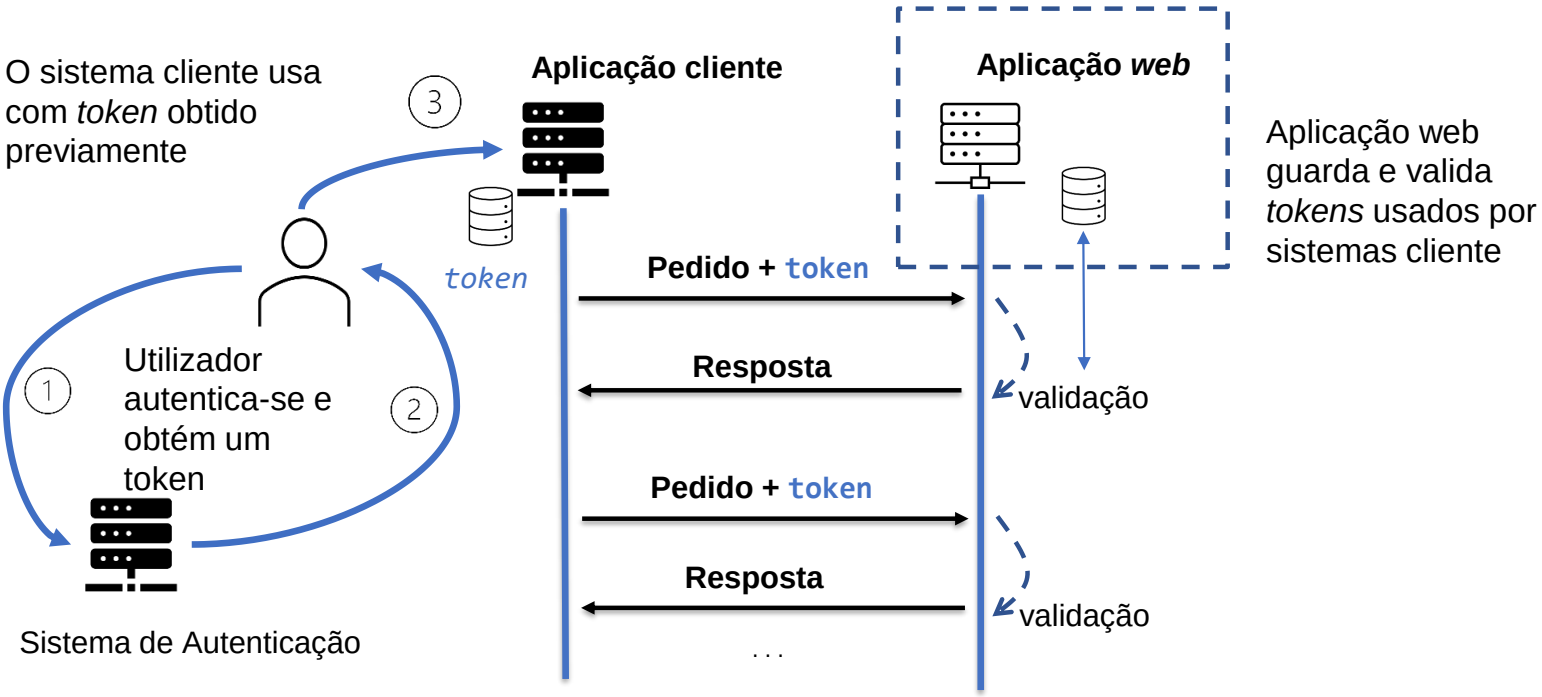


- Para evitar este problema, a opção `SameSite=Strict` garante que cookies marcados com esta opção não são usados em pedidos de *cross-site*

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

24

Exemplos de Autenticador: *token*



25

Autenticadores JWT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Header + "."
Payload + "."
Signature

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

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

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Base64 (header)

Base64 (payload)

K

HMAC

https://jwt.io/

26