Fundamentos de Segurança Informática (FSI)

2021/2022 - LEIC

Manuel Barbosa
mbb@fc.up.pt

Aula 12
Segurança Web: Modelo

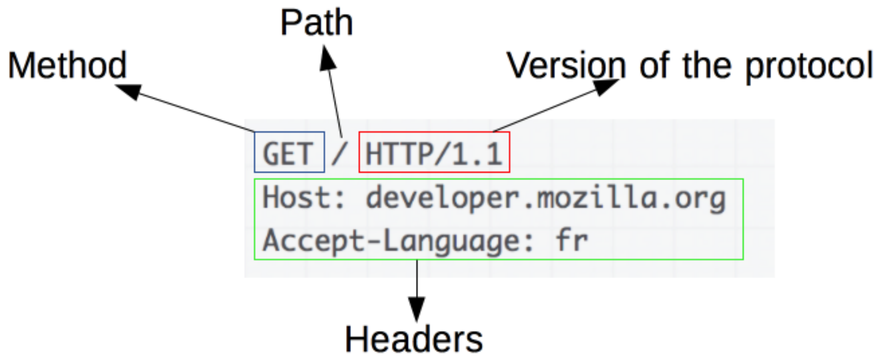
Protocolo HTTP

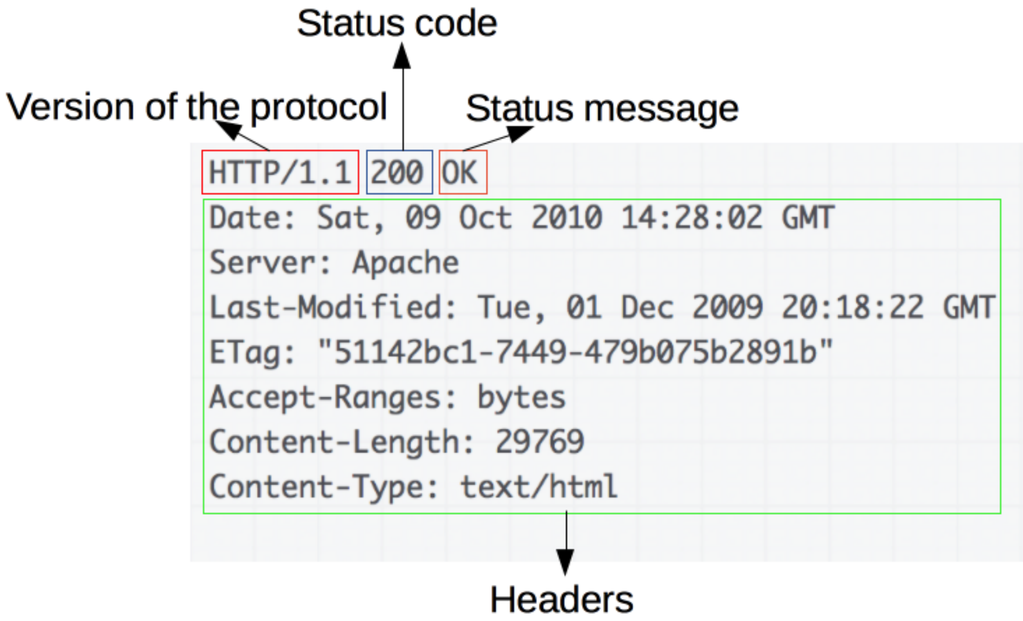
Protocolo HTTP

- Criado em 1989, permite a um cliente aceder a recursos armazenados num servidor
- Um recurso são identificados por uma localização uniforme (URL):
  - esquema (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)
- `https://sigarra.up.pt/fcup/en/cur_geral.cur_view?pv_curso_id=6041&pv_ano_lectivo=2021`

Protocolo HTTP

- Comunicação efetuada através de pedidos individuais:
  - GET: obter recurso numa URL específica
  - POST: criar um novo recurso numa URL específica
  - PUT: substituir representação de recurso existente com outro conteúdo
  - PATCH: alterar parte de um recurso
  - DELETE: apagar recurso numa URL específica





https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview

Protocolo HTTP

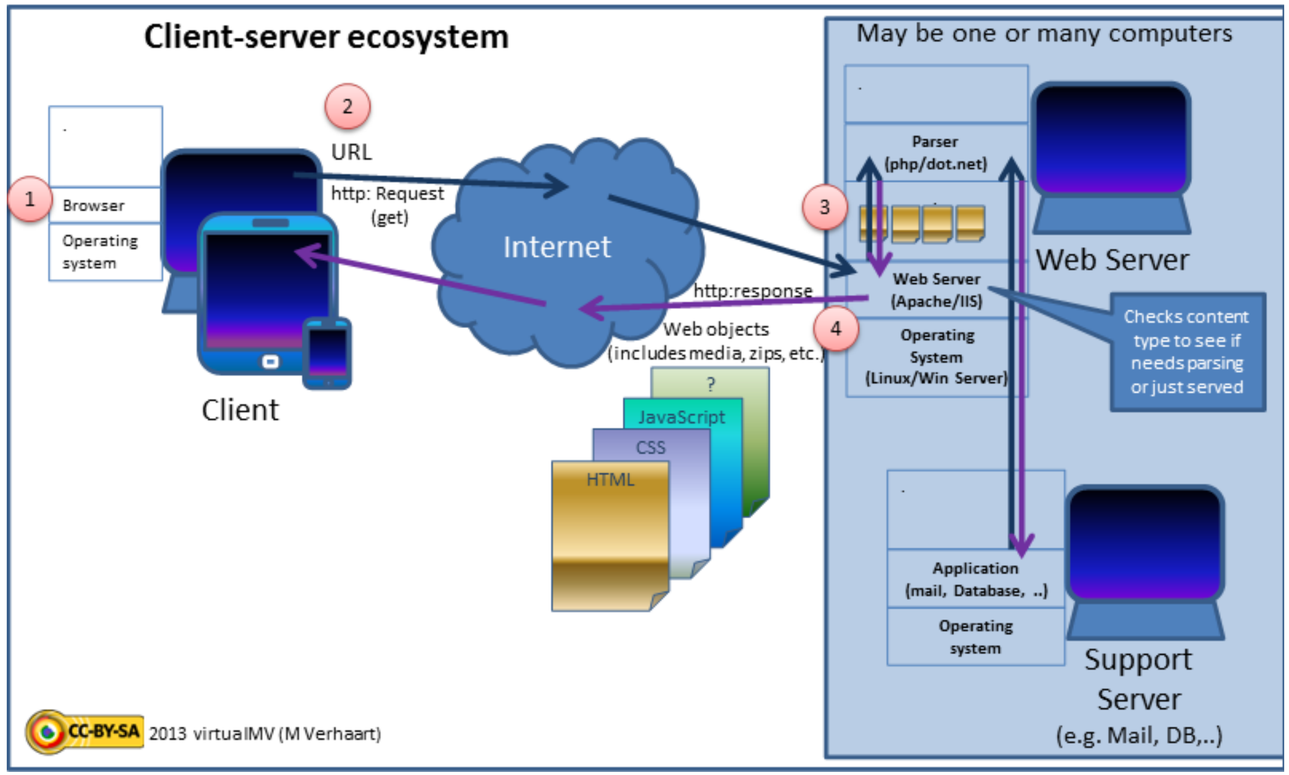
- Rapidamente a lógica foi adaptada/subvertida:
  - GET não devia alterar o estado do servidor, mas quase sempre tem side-effects
  - POST utilizado para quase tudo o que originalmente era previsto para PUT, PATCH e DELETE
  - As aplicações do servidor precisam de reconhecer pedidos relacionados: é preciso estado = *cookie*
    - servidor envia para browser, este armazena
    - sempre que uma URL é acedida no mesmo servidor a *cookie* é enviada
    - util para: gestão de sessões, personalização, rastreamento/profiling

Protocolo HTTP

- Recentemente o HTTP entrou em evolução rápida:
  - HTTP/2 (2015) => pipelining de pedidos + multiplexing + server push
  - HTTP/3 (draft de 2020) => abandona TCP e adota o QUIC da Google
- Nesta UC vamos ver apenas ataques/defesas clássicos, que ainda não tiram partido destas novas funcionalidades

Modelo de Execução

- Computações do lado do servidor:
  - bases de dados, gestão de sessões, personalização, etc.
- Computações do lado do cliente:
  - browsers = pequenos sistemas de virtualização
  - 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



https://www.wikiwand.com/en/Dynamic_web_page

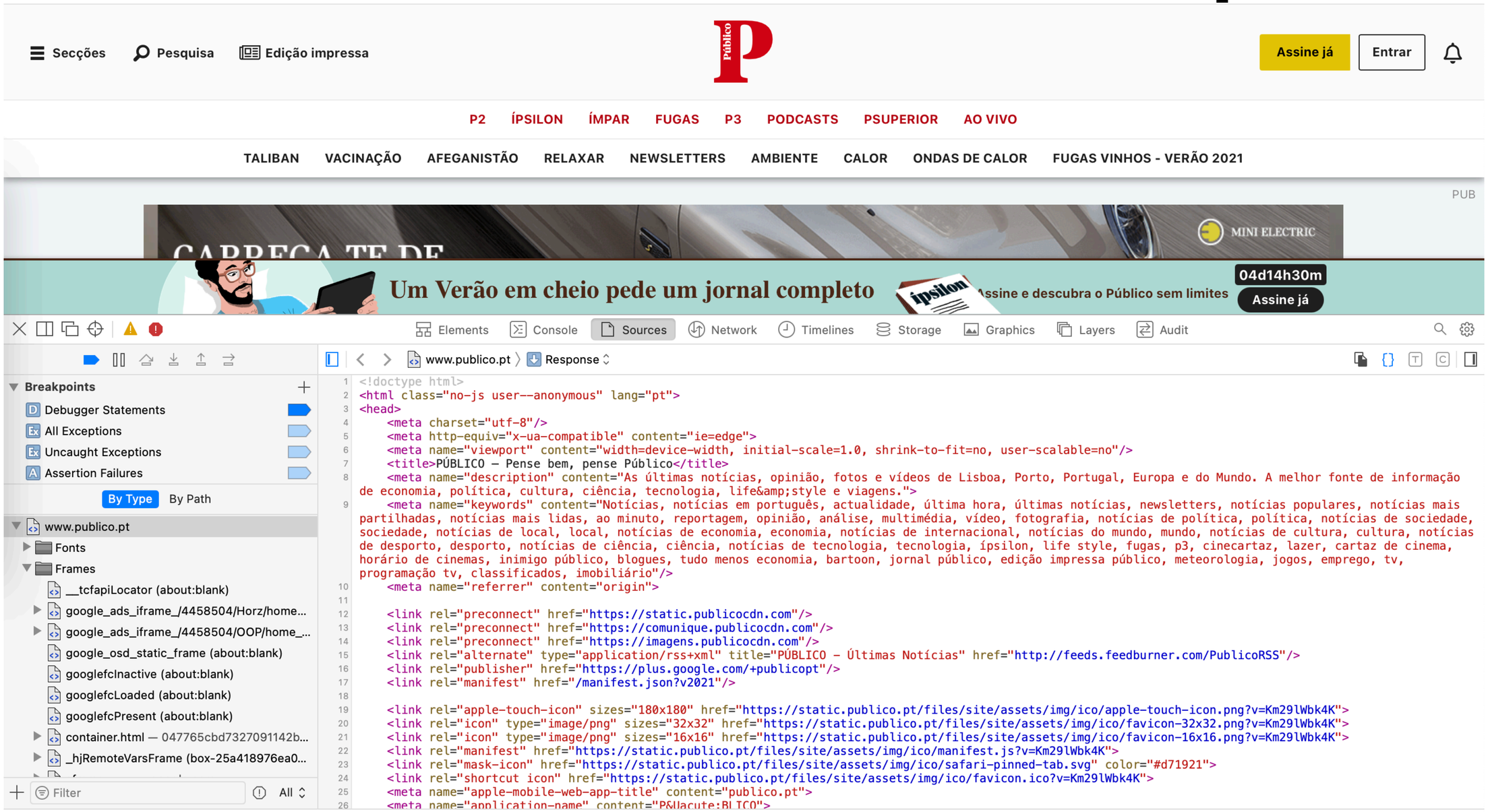
Modelo de Execução

- Uma janela do browser pode ter conteúdos de diferentes origens:
  - Frame (divisão rígida visível), iFrame (objeto flutuante/imbricado num documento)
  - Porquê utilizar (i)frames?
    - delegar uma área do ecrã para outra origem
    - aproveitar proteção do browser direcionada para isolamento de frames
    - isolamento permite a página *mãe* funcionar mesmo que frame falhe
- O código JavaScript pode ler e alterar o estado de uma página com o DOM:
  - interface orientada ao objecto, que representa toda a página de forma hierárquica

Os sites atuais são complexos!

- Exercício:
  - escolher site, e.g. jornal, preferido e contar
  - número de recursos
  - número de fontes diferentes e os seus países
  - quantas dessas fontes são controladas pelo site principal
  - número de cookies

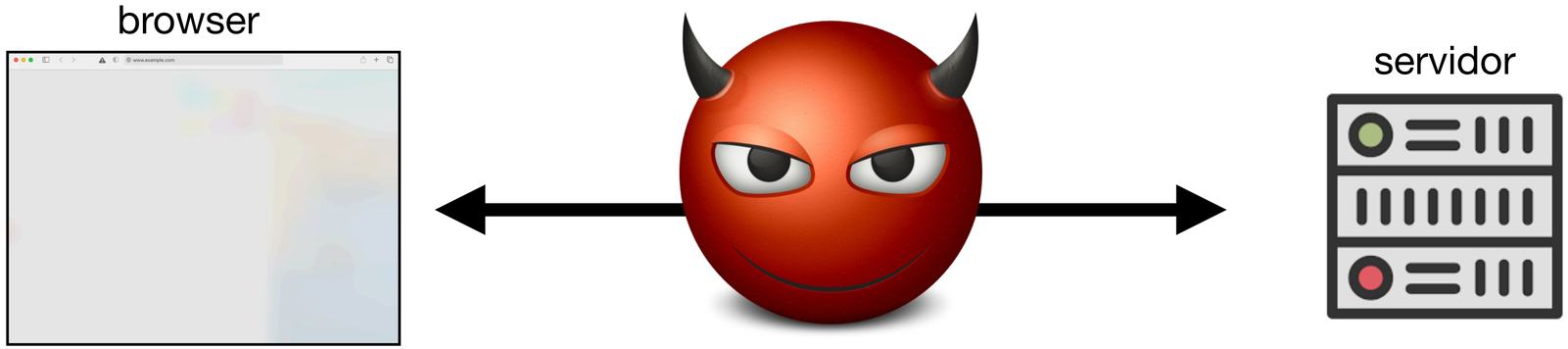
Os sites atuais são complexos!



Modelos de Ataque

Atacante externo/rede

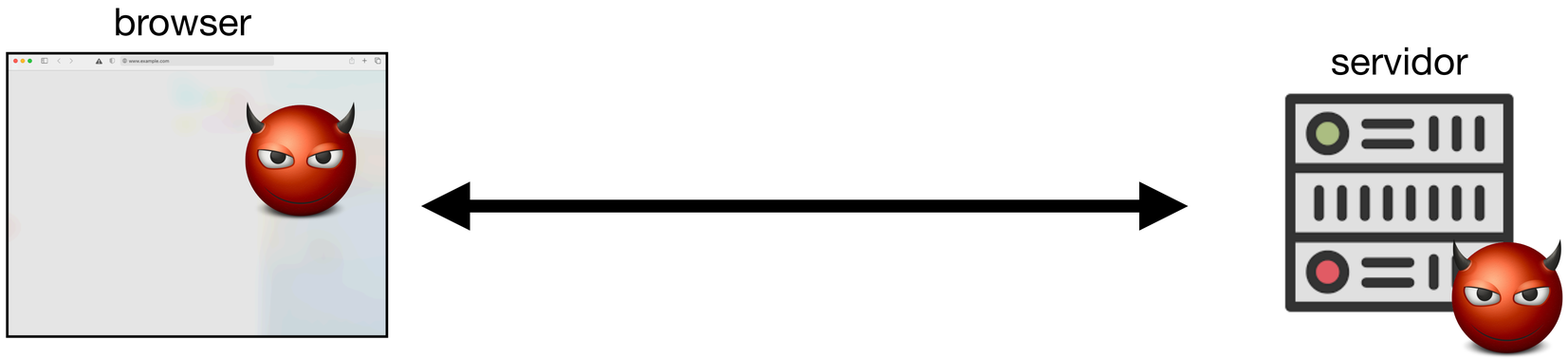
- Adversário que controla apenas o meio de comunicação



- Abordaremos esta classe de adversários quando falarmos sobre segurança de redes

Atacante interno/web

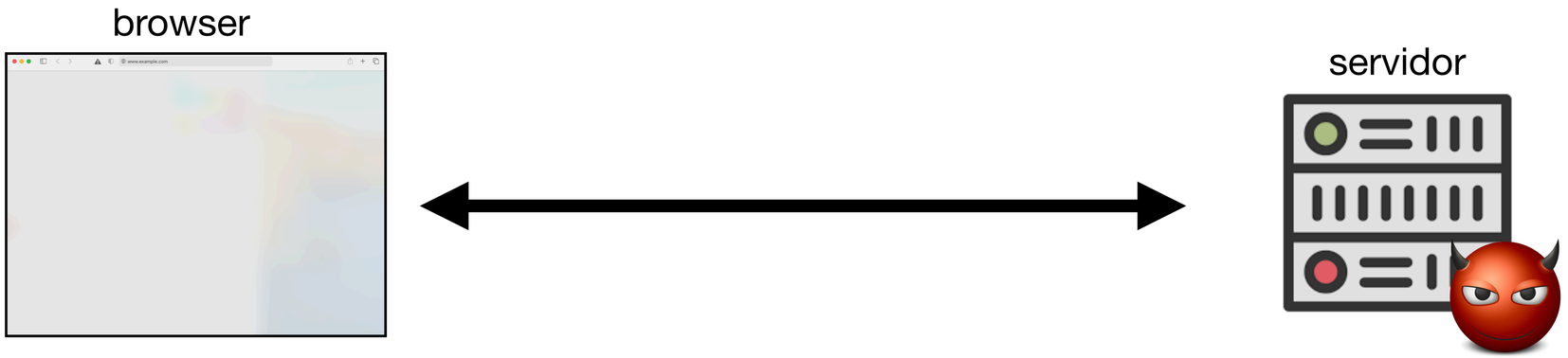
- Adversário que controla parte da aplicação web



- Existem diversas variantes deste modelo que iremos ver a seguir

Atacante web

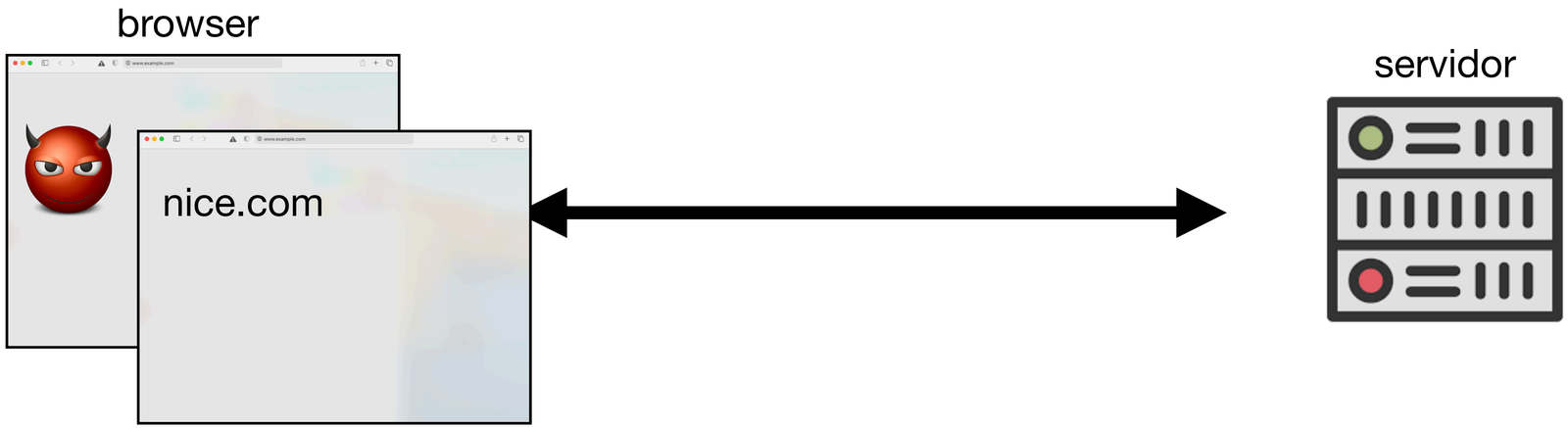
- Adversário que controla servidor: impedir abuso da máquina cliente



- Atacante que controla cliente: impedir abuso dos recursos no servidor

Atacante interno/web

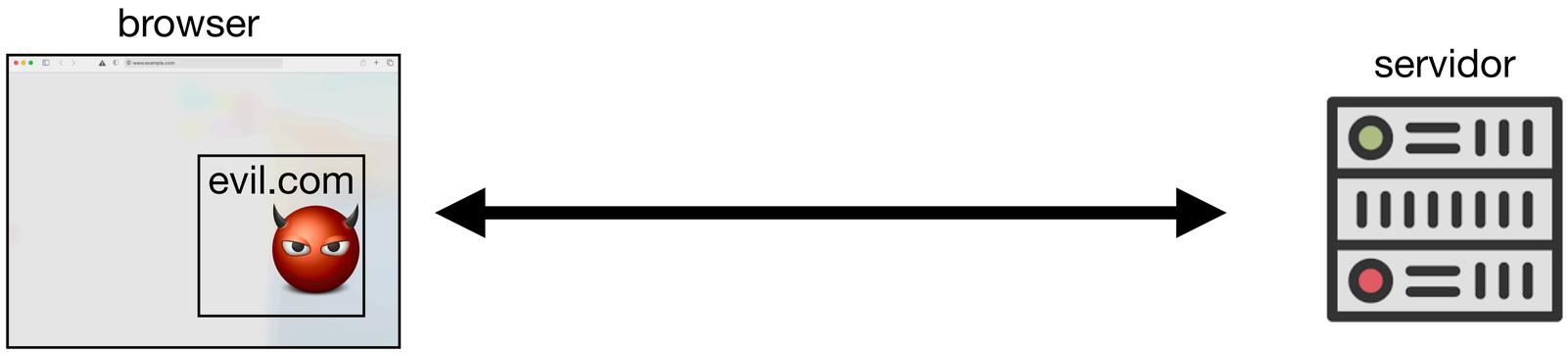
- Adversário que controla uma página no cliente!



- Impedir que página A interfira ou observe página B (e tudo o resto)
- Em que podemos confiar neste caso?

Atacante interno/web

- Adversário que controla *parte* de uma página no cliente!

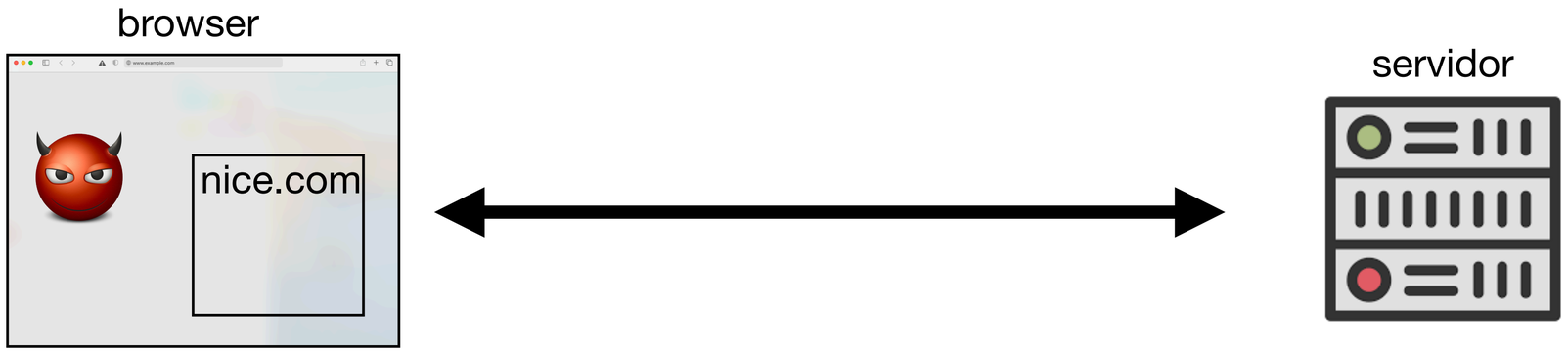


- Impedir que objeto embebido na página A interfira com página A (e tudo o resto)

- Em que podemos confiar neste caso?

Atacante interno/web

- Adversário que controla *parte* de uma página no cliente!



- Impedir que página A interfira com objeto embebido na página A (e tudo o resto)

- Em que podemos confiar neste caso?

Modelo de Segurança

Analogia browser/sistema operativo

Processos	Páginas
Ficheiros	Cookies
Sockets/TCP	Fetch/HTTP
Sub-processos	Frames/iFrames

- Isolamento:
  - **Same Origin Policy (SOP)**
- Origem: contexto de isolamento que corresponde a fronteira de confiança na Web
- SOP:
  - 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

Same Origin Policy (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
- mensagens: frames podem comunicar entre si!

```
function receiveMessage(event){
if (event.origin !== "http://example.com")
    return; ...
}
```

- comunicações com servidor: mais subtil … (ver slides seguintes)
- cookies: apenas enviados pelo browser para servidores com a mesma origem que as criou (ver nuances mais à frente)

SOP para pedidos ao servidor

- Uma página/frame pode fazer pedidos HTTP fora do seu domínio
  - escrita geralmente permitida: permite enviar/expor dados a serviços!
  - o embedding de recursos de outras origens é geralmente permitido
  - os dados contidos na resposta:
    - podem ser processados nativamente pelo browser (e.g., criar frame)
    - não podem ser analisados programaticamente (por princípio)
    - mas o embedding expõe alguma informação

SOP para pedidos ao servidor

- Exemplos:
  - código HTML:
    - podemos criar frames com código HTML de outras origens
    - não podemos inspecionar ou modificar o conteúdo da frame
  - scripts JavaScript:
    - podemos obter scripts de outras origens
    - podemos executar scripts de outras origens (no contexto da nossa origem!)
    - não deve ser possível a JavaScript executado no contexto de uma origem inspecionar/ manipular código JavaScript carregado de outra origem
      (Certas plataformas permitem toString, portanto esta restrição é duvidosa)

SOP para pedidos ao servidor

- Mais exemplos:
  - imagens:
    - browser faz rendering
    - SOP proíbe ver pixeis, mas revela tamanho
  - para CCS e fontes a regras são semelhantes

https://arxiv.org/pdf/2004.00939.pdf

SOP para pedidos ao servidor

- A documentação disponível online nem sempre é totalmente esclarecedora/ consistente
  - nem sempre é claro o que é que um site malicioso pode observar quando faz embedding de um objeto
  - servidor não pode assumir que recursos públicos enviados para a página são inócuos: podem sempre revelar informação sensível sobre o estado do servidor
    ==> e.g., fotografia pessoal de tamanho diferente se logged in

https://www.netsparker.com/whitepaper-same-origin-policy/

As an example, when evaluate the behavior of an **ambiguous image**, we must remember the rules that Same-origin Policy defines:

1. Each site has its own resources like cookies, DOM, and Javascript namespace.
2. Each page takes its origin from its URL (normally schema, domain, and port).
3. Scripts run in the context of the origin which they are loaded. It does not matter where you load it from, only where it is finally executed.
4. Many resources, like media and images, are passive resources. They **do not have access** to objects and resources in the context they were loaded.

Given these rules, **we can assume** that a site with origin *A*:

1. Can load a script from origin *B*, but it works in *A*'s context
2. Cannot reach the raw content and source code of the script
3. Can load CSS from the origin *B*
4. Cannot reach the raw text of the CSS files in origin *B*
5. Can load a page from origin *B* by iframe
6. Cannot reach the DOM of the iframe loaded from origin *B*
7. Can load an image from origin *B*
8. Cannot reach the bits of the image loaded from origin *B*
9. Can play a video from origin *B*
10. Cannot capture the images of the video loaded from origin *B*

CORS

- Cross-Origin Resource Sharing: permite relaxar os pedidos cross-origin que são permitidos (em particular pedidos construídos dinamicamente a partir de JavaScript, para acesso aos recursos)
  - Pedidos simples de site A de recursos no servidor B:
    - não devem causar side-effects no servidor B
    - browser faz pedido e verifica se resposta admite que o código em A pode aceder aos recursos provenientes de B
    - o servidor B pode **permitir** mais casos de uso através do atributo

```
Access-Control-Allow-Origin
```

CORS

- Cross-Origin Resource Sharing: permite relaxar os pedidos cross-origin que são permitidos (em particular pedidos construídos dinamicamente a partir de JavaScript, para acesso aos recursos)
  - Pedidos *pre-flighted* de site A de recursos no servidor B:
    - podem causar side-effects no servidor B
    - browser faz pedido **dummy** e verifica se resposta admite que o código em A pode aceder aos recursos provenientes de B
    - o servidor B pode **permitir** mais casos de uso através do atributo

      `Access-Control-Allow-Origin`

    - se o acesso for permitido então browser faz pedido real

SOP para cookies: definir

- A definição de origem é diferente: domínio e path, o esquema é opcional
- Uma página pode **definir** uma cookie para (write-up):
  - o seu domínio
  - domínios hierarquicamente superiores (exceto para *sufixos públicos*)

SOP para cookies

PUBLIC SUFFIX LIST

LEARN MORE | THE LIST | SUBMIT AMENDMENTS

Return to the Public Suffix List homepage

A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are `.com`, `.co.uk` and `pvt.k12.ma.us`. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is an initiative of Mozilla, but is maintained as a community resource. It is available for use in any software, but was originally created to meet the needs of browser manufacturers. It allows browsers to, for example:

- Avoid privacy-damaging "supercookies" being set for high-level domain name suffixes
- Highlight the most important part of a domain name in the user interface
- Accurately sort history entries by site

We maintain a fuller (although not exhaustive) list of what people are using it for. If you are using it for something else, you are encouraged to tell us, because it helps us to assess the potential impact of changes. For that, you can use the psl-discuss mailing list, where we consider issues related to the maintenance, format and semantics of the list. Note: please do not use this mailing list to request amendments to the PSL's data.

It is in the interest of Internet registries to see that their section of the list is up to date. If it is not, their customers may have trouble setting cookies, or data about their sites may display sub-optimally. So we encourage them to maintain their section of the list by submitting amendments.

```
// ===BEGIN ICANN DOMAINS===

// ac : https://en.wikipedia.org/wiki/.ac
ac
com.ac
edu.ac
gov.ac
net.ac
mil.ac
org.ac

// ad : https://en.wikipedia.org/wiki/.ad
ad
nom.ad

// ae : https://en.wikipedia.org/wiki/.ae
// see also: "Domain Name Eligibility Policy" at http://www.aeda.ae/eng/aepolicy.php
ae
co.ae
net.ae
org.ae
sch.ae
ac.ae
gov.ae
mil.ae

// aero : see https://www.information.aero/index.php?id=66
aero
accident-investigation.aero
accident-prevention.aero
aerobatic.aero
aeroclub.aero
aerodrome.aero
agents.aero
aircraft.aero
airline.aero
airport.aero
air-surveillance.aero
airtraffic.aero
air-traffic-control.aero
ambulance.aero
amusement.aero
association.aero
author.aero
ballooning.aero
broker.aero
```

SOP para cookies: enviar

- Tradicionalmente os browsers enviavam todas as cookies no contexto de uma URL (diretamente no pedido):
  - se o domínio da cookie for um sufixo do domínio da URL
  - **E** se a path da cookie for um prefixo da path da URL
- Os novos browsers funcionam assim apenas se `SameSite=None` (à frente)

SOP para cookies: enviar

	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

Figura: Nadia Heninger

SOP para cookies: enviar

- SameSite = Strict
  - apenas envia cookie quando o pedido tem a mesma origem que a top-level frame
- SameSite = Lax
  - distingue alguns pedidos (e.g., links explícitos) e abre exceções *cross domain* para o envio de cookies
- Secure cookies
  - apenas enviadas por https

SOP para cookies

- Será que o SOP de cookies (usando path) dá um controlo mais fino do que o SOP de DOM?
    - Não! Na realidade o acesso em regime DOM existe sempre
      E.g. se mesmo domínio mas path diferente, então são acessíveis via DOM
- Outro exemplo:
    - se uma página cria uma frame com outra origem
    - as cookies associadas à frame não são acessíveis à página mãe
        - Estarão protegidas? Depende de quem as pretendemos esconder
          E.g. se transmitidas em canal aberto serão visíveis na rede
- Um outro exemplo:
    - página do banco executa java script de Google analytics
    - o java script executa debaixo do contexto da página do banco!
    - o java script pode ler a variável document.cookie! (Session Hijacking!)
    - para evitar este problema pode definir-se uma cookie como `HTTPOnly: não acessível de JavaScript`