Cross-Site Script Inclusion: um estudo das estratégias de mitigação e atual prevalência da vulnerabilidade em navegadores

Henrique Curi de Miranda[1,2], **Henrique Arcoverde**[2], **Renan Villela Oliveira**[1], **Luiz H. A. Correia**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Caixa Postal 3037 – 37.200-900 – Lavras – MG

[2]Tempest Security Intelligence
Rua da Alfândega, 35 – Loja 216A – 1º Piso – Recife – PE

henriquecuri1@proton.me, henrique.arcoverde@tempest.com.br,
renanvillela@protonmail.com, lcorreia@ufla.br

Abstract. *The Cross-Site Script Inclusion (XSSi) is a little-known vulnerability that exploits the attachment of cookies in cross-origin requests. This work presents an evaluation of the behavior of the SameSite attribute in cookies across recent versions of different browsers and its impact on the exploitation of Cross-Site Script Inclusion. Experiments were conducted to verify the occurrence of cookie attachment in cross-origin requests for various browsers. The main strategy proposed to mitigate the XSSi vulnerability is the correct configuration of the SameSite attribute by the application developer. The results show that this strategy for combating Cross-Site Script Inclusion is effective, as it presents fewer points of failure. Additionally, it was concluded that there is no standard implementation of the same-origin policy and SameSite by current browsers, with divergences in behavior regarding cookie attachment in cross-origin requests.*

Resumo. *A Cross-Site Script Inclusion ou XSSi é uma vulnerabilidade pouco conhecida e que abusa da anexação de cookies em requisições de origem cruzada. Este trabalho apresenta uma avaliação do comportamento do atributo SameSite dos cookies, em versões recentes de diferentes navegadores, e do seu impacto na exploração da Cross-Site Script Inclusion. Experimentos foram realizados verificando a ocorrência da anexação de cookies em requisições cruzadas para diversos navegadores. A principal estratégia proposta para mitigação da vulnerabilidade XSSi é a correta configuração da SameSite pelo desenvolvedor da aplicação. Resultados mostram que essa estratégia para combate da Cross-Site Script Inclusion é eficiente por apresentar menos pontos de falha. Em adição, foi concluído que não existe um padrão de implementação da política de mesma origem e da SameSite por parte dos atuais navegadores, havendo divergência de comportamento quanto a anexação de cookies em requisições de origem cruzada.*

1. INTRODUÇÃO

As aplicações web frequentemente armazenam e lidam com dados sensíveis que devem ser protegidos contra o acesso de agentes maliciosos. A utilização dos dados é parte essencial para as operações dessas aplicações. As funções que tratam esses dados e os seus mecanismos de controle de acesso dos usuários e de suas permissões, devem garantir a proteção contra acessos não autorizados. É essencial que essas funções tenham sido desenvolvidas de forma correta, de maneira a não gerar falhas de segurança, impedindo que agentes maliciosos se beneficiem de acessos não autorizados.

Uma das possíveis falhas de segurança consiste na vulnerabilidade denominada *Cross-Site Script Inclusion* (*XSSi*), que é baseada no abuso da Política de Mesma Origem e na anexação de *cookies* em requisições de origem cruzada. A XSSi permite a inclusão de dados sensíveis em contextos de diferentes origens, resultando em uma possível manipulação a partir de uma página maliciosa [HAILPERIN and RUEF 2016]. Uma das maneiras mais populares de se tratar a segurança da informação é um modelo baseado em três pilares: disponibilidade, integridade e confidencialidade. A vulnerabilidade XSSi resulta em violação da confidencialidade dos dados armazenados na aplicação, uma vez que pode permitir acesso não autorizado.

De maneira geral, a *Cross-Site Script Inclusion* consiste em uma classe de vulnerabilidades que ocorrem de maneira similar: uma página maliciosa, ao ser acessada, leva o navegador a realizar uma requisição de origem cruzada de maneira implícita, tendo por alvo uma aplicação legítima, e com o objetivo de inclusão de recursos no contexto da página maliciosa. A anexação de *cookies* é essencial neste contexto, uma vez que garante à página maliciosa acesso aos dados privados, abusando de um mecanismo de controle de permissões. Uma vez que o recurso é incluído no contexto malicioso, o atacante é capaz de manipular esses dados, dependendo da maneira como a página maliciosa foi desenvolvida, sendo possível o atacante acessar esses dados, a partir de manipulação de variáveis no contexto malicioso e realização de requisições para um domínio de sua posse.

No contexto dos navegadores, a Política de Mesma Origem (SOP - *Same-origin Policy*) é um mecanismo de suma importância para a segurança dos dados [Mozilla 2023b]. Com a SOP, as páginas carregadas, bem como os dados referentes ao seu contexto, são organizados a partir da origem definida por três itens: protocolo, domínio e porta. Portanto, duas páginas serão consideradas como pertencentes a uma mesma origem caso tenham sido geradas por um mesmo protocolo, um mesmo domínio e uma mesma porta. A Política de Mesma Origem determina um isolamento de contexto entre páginas de origens distintas, impedindo o acesso e a interação entre dados referentes a cada origem. Qualquer requisição gerada por uma origem, e que possui como destino uma origem diferente, é denominada ”requisição de origem cruzada”.

O controle de acesso é essencial para tratar da segurança de dados, sendo necessário segregação entre usuários com e sem permissão de acesso. Uma das maneiras de identificar usuários de uma aplicação, bem como suas respectivas permissões, são os *cookies*: pequenas cadeias de texto armazenadas no navegador do usuário [Mozilla 2023a]. A utilização dos *cookies* consiste na sua anexação em requisições HTTP, sendo recebidos e interpretados pelo servidor destinatário, parte importante para a implementação do conceito de sessão em um protocolo concebido originalmente sem este recurso [Kurose and Ross 2009].

Os *cookies* podem receber diferentes atributos, cujo valor é definido de maneira explícita ou implícita no momento que a aplicação servidora os gera. Esses atributos modificam o comportamento do navegador, influenciando na anexação de *cookies* em diferentes requisições. Um desses atributos é o *SameSite*, desenvolvido pela Google e Mozilla em abril de 2016 e implementado em seus respectivos navegadores (Chrome e Firefox) posteriormente [WEST and GOODWIN 2016].

A *SameSite* determina o comportamento do *cookie* quanto a sua anexação em requisições de origem cruzada, podendo receber três valores: *Strict*, *Lax*, *None*. Respectivamente, cada valor teoricamente determina o seguinte comportamento: anexação em nenhuma requisição cruzada, anexação em requisições cruzadas provenientes de navegação por URL, anexação em requisições de origem cruzada. A não declaração explícita do valor do atributo, determina o comportamento referente ao *Lax*.

A análise dos trabalhos relacionados à *Cross-Site Script Inclusion* revelam um período de relativo hiato, tendo o cenário atual da vulnerabilidade sido pouco explorado na literatura, apesar da implementação de mudanças em mecanismos diretamente relacionados com o problema. O reflexo disso pode ser percebido ao verificar os principais materiais online a respeito do assunto, que muitas vezes não englobam essas mudanças, refletindo no trabalho de profissionais ligados ao desenvolvimento e manutenção de serviços web. Além disso, percebe-se uma deficiência quanto à avaliação da vulnerabilidade XSSi em cenários reais e que considerem os atuais mecanismos empregados nos navegadores mais recentes.

Este trabalho apresenta os impactos da implementação do atributo *SameSite* quanto à exploração da *Cross-Site Script Inclusion*. O objetivo é observar o comportamento de exploração dessa vulnerabilidade, em navegadores atualizados, e o impacto da implementação de diferentes valores para o atributo *SameSite* do *cookie* gerado quanto à anexação deste na requisição cruzada. A hipótese consiste em uma divergência de comportamento entre os diferentes navegadores testados, bem como da ocorrência e não ocorrência da vulnerabilidade, a depender do valor atribuído para a *SameSite*, de maneira explícita ou não. Foram simuladas implementações reais a respeito do comportamento do atributo em versões recentes de diferentes navegadores, comparando-se a diferença do comportamento padrão quanto ao atributo *SameSite*. Resultados mostraram que certos navegadores são mais suscetíveis que outros quanto à exploração da vulnerabilidade, levando também ao usuário a possibilidade de um papel mais ativo para mitigação da *Cross-Site Script Inclusion*. A avaliação demonstrou que a principal estratégia de mitigação da vulnerabilidade é baseada em uma configuração adequada do atributo *SameSite*.

A organização deste artigo respeita o que se descreve a seguir. A Seção 2 consiste na apresentação e descrição de trabalhos anteriormente realizados por outros pesquisadores. A Seção 3 descreve o ambiente de testes utilizado para o estudo prático conduzido. A Seção 4 consiste na descrição individual de cada caso implementado e sua respectiva exploração. A Seção 5 apresenta a avaliação dos resultados obtidos com os testes. A Seção 6 contém as conclusões obtidas e sugestões de trabalhos futuros.

2. Trabalhos Relacionados

Ao longo dos anos, diversos pesquisadores realizaram trabalhos a respeito da *Cross-Site Script Inclusion*, sendo apresentadas técnicas tanto de exploração quanto de mitigação para o problema.

O primeiro trabalho, aquele que apresentou inicialmente a ocorrência da vulnerabilidade, foi publicado por Jeremiah Grossman no seu blog pessoal no ano de 2006 [GROSSMAN 2006]. Grossman apresentou a exploração de uma vulnerabilidade na aplicação Gmail, tendo por alvo um vetor JSON preenchido dinamicamente com a lista de contatos do usuário identificado através do *cookie* de sessão. O autor demonstrou a inclusão deste arquivo no contexto de uma página de origem cruzada, a partir do acesso do usuário a uma página maliciosa, abusando da anexação do *cookie* em requisição cruzada, consequentemente levando também ao acesso não autorizado de dados sensíveis.

Grossman não definiu sua descoberta como uma nova vulnerabilidade, sendo este fato atribuído a Cristoph Kern que, junto de outros autores, publicou o livro "*Foundations of Security What Every Programmer Needs to Know*" [KERN et al. 2007]. Na obra, é definido o termo *Cross-Site Script Inclusion*, sendo formalizada a sua definição para essa classe de vulnerabilidades através de uma subseção dedicada ao assunto.

A obra de Kern também define estratégias de mitigação para a vulnerabilidade em seu livro. A estratégia principal apresentada por Kern consiste na utilização de um *Token* de ação imprevisível para uma página maliciosa, gerado quando necessário a interação com um arquivo JavaScript dinâmico, estratégia esta também apresentada pela obra como uma estratégia para prevenção da *Cross-Site Request Forgery* e comumente implementada até hoje. Essa estratégia apresentada por Kern deve estar aplicada para todos os arquivos sensíveis e possivelmente vulneráveis a *Cross-Site Script Inclusion* de uma aplicação, gerando múltiplos pontos que requerem atenção por parte dos desenvolvedores.

No ano de 2015, o pesquisador Takeshi Terada apresentou a possibilidade de inclusão de um arquivo CSV em um contexto de origem cruzada [TERADA and BUSSAN 2015]. Com isso, surgiu o termo "*Non-Script XSSi*", sendo demonstrada a possibilidade de exploração da vulnerabilidade em outros tipos de arquivos além de *scripts*.

Sebastian Lekies, também no ano de 2015, realizou um estudo específico para os casos de aplicações utilizando código JavaScript dinamicamente gerado [LEKIES et al. 2015]. Em uma busca realizada neste período, nos 150 sites classificados pelo *Alexa Top Websites*, aos quais foi possível a realização de cadastro, foram realizadas tentativas de exploração da *Cross-Site Script Inclusion* nestes arquivos dinâmicos. O trabalho de Lekies demonstrou, de maneira prática, a prevalência do problema em aplicações web populares, apontando estratégias de mitigação: separação de dados dinâmicos do código JavaScript estático, utilização de caminhos dinâmicos para os arquivos e checagem de origem de requisições de inclusão. Os resultados obtidos por Lekies estão contidos na Tabela 1.

No ano de 2018, Franken et al. publicaram uma pesquisa na qual houve a avaliação do comportamento dos *cookies* de terceiros considerando sete navegadores diferentes. Os *cookies* de terceiros foram amplamente utilizados no passado para rastreamento de usuários entre diferentes origens [FRANKEN et al. 2018]. Resultados mostraram a

Tabela 1. Resultados obtidos por Lekies

Fonte: Usenix - The Unexpected Dangers of Dynamic JavaScript

	Nº de Domínios	**Explorável**
Scripts dinâmicos baseados em *cookies*	49	40
Identificadores únicos contidos	34	28
Outros dados pessoais contidos	15	11
Tokens CSRF ou de autenticação contidos	7	4

existência de uma divergência de implementação quanto ao comportamento de anexação dos *cookies* em requisições de origem cruzada entre os navegadores analisados.

A implementação da *SameSite* pela maioria dos navegadores mudou de maneira drástica a dinâmica de requisições de origem cruzada de maneira geral. Nesse sentido, o trabalho apresentado por Khodayari e Pellegrino em 2022, revelou de maneira ampla o impacto da adoção do atributo pelos navegadores [KHODAYARI and PELLEGRINO 2022]. O estudo conduzido realizou uma avaliação longitudinal em 500 mil sites, sendo observado que 18.640 (3,7%) destes utilizavam uma política *SameSite* permissiva através do valor *None*, de maneira a contornar as restrições impostas pelo atributo. Os pesquisadores também observaram uma tendência de maior permissibilidade entre os sites mais populares, sendo que 18% entre os mil mais populares implementaram uma política permissiva. Khodayari e Pellegrino também observaram uma divergência de implementação do atributo *SameSite* entre diferentes navegadores, reforçando uma ausência de padronização.

A Figura 1 resume, através de uma linha do tempo, os trabalhos anteriores. Também é destacada a principal contribuição de cada pesquisa para o contexto desta.

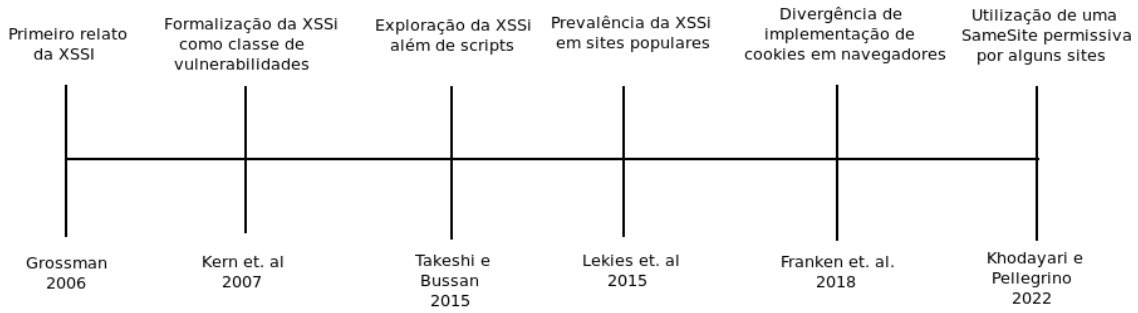

Figura 1. Linha do tempo de trabalhos anteriores

A partir dos trabalhos anteriores e das conclusões expostas pelos pesquisadores, foi possível traçar hipóteses a serem testadas para este trabalho. A Seção seguinte apresenta em maiores detalhes o ambiente de testes desenvolvido, além das propostas iniciais a serem testadas.

3. Ambiente de Testes

Para o estudo objetivo deste trabalho, foi desenvolvido um ambiente de testes consistindo em: um servidor hospedando uma aplicação vulnerável, um servidor hospedando uma aplicação maliciosa e uma máquina cliente hospedando navegadores. Em resumo, as características de cada elemento do ambiente de testes estão contidas na Tabela 2.

Tabela 2. Características do ambiente de teste

	Servidor Vulnerável	**Servidor Malicioso**	**Cliente**
Sistema Operacional	*Debian* 12 *Kernel* 6.1.85-1	*Debian* 12 *Kernel* 6.1.85-1	*Windows 11 Enterprise Evaluation 22H2*
Aplicação	*Apache* 2.4.59	*Apache* 2.4.59	Navegadores diversos
Hospedagem	*AWS Lightsail*	*AWS Lightsail*	*VirtualBox*

Para os servidores, foi utilizado o serviço ”AWS Lightsail”e o serviço de DNS ”DuckDNS”, possibilitando o acesso via internet e provendo um domínio atrelado. Posteriormente, foi implementado o certificado e o acesso via HTTPS, empregando a ferramenta ”Certbot”. A máquina cliente foi implementada como uma máquina virtual hospedada localmente, através do software ”VirtualBox”, e com acesso à internet.

O servidor vulnerável utiliza o sistema operacional Debian 12 na versão 6.1.85-1 do *kernel* Linux, hospedando um servidor web Apache na versão 2.4.59. A aplicação consiste em uma página inicial de autenticação que, ao serem fornecidas credenciais corretas, gera um *cookie* e o armazena no navegador, permitido acesso a uma área protegida. A área protegida consiste em um menu com *hyperlinks* direcionados para os seguintes arquivos: um arquivo JavaScript, uma imagem e um arquivo JSONP (JSON *with padding*). Foram definidas as seguintes regras: uma requisição direcionada aos arquivos protegidos com anexação do *cookie*, gera um código de resposta ”200” e acesso ao recurso, enquanto que uma requisição sem anexação do *cookie* gera um código de resposta ”403” e o acesso ao recurso não é permitido. As regras foram definidas através da implementação do arquivo de configuração ”*.htaccess*” para o servidor web, sendo a seguinte:

```
<Files "example.jpg">
    RewriteEngine On
    RewriteCond %{HTTP_COOKIE} !authenticated=true [NC]
    RewriteRule ^ - [F]
</Files>

<Files "example.js">
    RewriteEngine On
    RewriteCond %{HTTP_COOKIE} !authenticated=true [NC]
    RewriteRule ^ - [F]
</Files>

<Files "example.jsonp">
    RewriteEngine On
    RewriteCond %{HTTP_COOKIE} !authenticated=true [NC]
    RewriteRule ^ - [F]
</Files>
```

O servidor malicioso também utiliza o sistema operacional Debian 12 na versão 6.1.85-1 do *kernel* Linux, hospedando um servidor web Apache na versão 2.4.59. A aplicação consiste em um menu com *hyperlinks* direcionados para três páginas maliciosas, cada uma desenvolvida com o intuito de explorar um respectivo caso implementado na aplicação vulnerável, através de requisições de origem cruzada a partir de *tags* do HTML.

A versão do Debian e do Apache foram providas de acordo com o padrão disponibilizado pelo serviço ”AWS Lightsail” no momento da realização dos testes. Por se tratar de uma vulnerabilidade que ocorre no navegador, a *Cross-Site Script Inclusion* não depende das versões de sistema operacional e do servidor web nas quais as páginas são

hospedadas, dependendo apenas do navegador utilizado pela vítima e da maneira como os *cookies* são gerados.

A máquina cliente utiliza por sistema operacional o "Windows 11 Enterprise Evaluation 22H2", sendo a versão mais recente disponibilizada pela Microsoft no momento dos testes. Foram utilizados, na versão mais recente disponível em Janeiro de 2024, os seguintes navegadores baseados em Chromium e Firefox, além das próprias bases: Avast Secure Browser, Brave, Chrome, Chromium, Edge, Firefox, Librewolf, Opera, OperaGX, Tor Browser e Vivaldi. Para cada navegador foi utilizado o fluxo padrão de instalação, sem qualquer configuração adicional durante o processo ou após. A Tabela 3 (Seção 5) apresenta o número de versão de cada navegador analisado para este trabalho.

Neste ambiente de testes, a interação entre seus componentes ocorre da seguinte forma: a máquina cliente é utilizada para acessar a aplicação vulnerável através de um navegador e realizar autenticação, gerando assim o *cookie* que permite acesso à área protegida da aplicação. Após a geração do *cookie*, ainda com a máquina cliente, é realizado acesso à aplicação maliciosa em cada caso implementado, sendo verificado o comportamento através da captura dos pacotes HTTP de requisição e resposta. Foi utilizada a versão 1.1 do protocolo HTTP. A Figura 2 exemplifica o modelo de interação entre os componentes quando ocorre a exploração da vulnerabilidade.

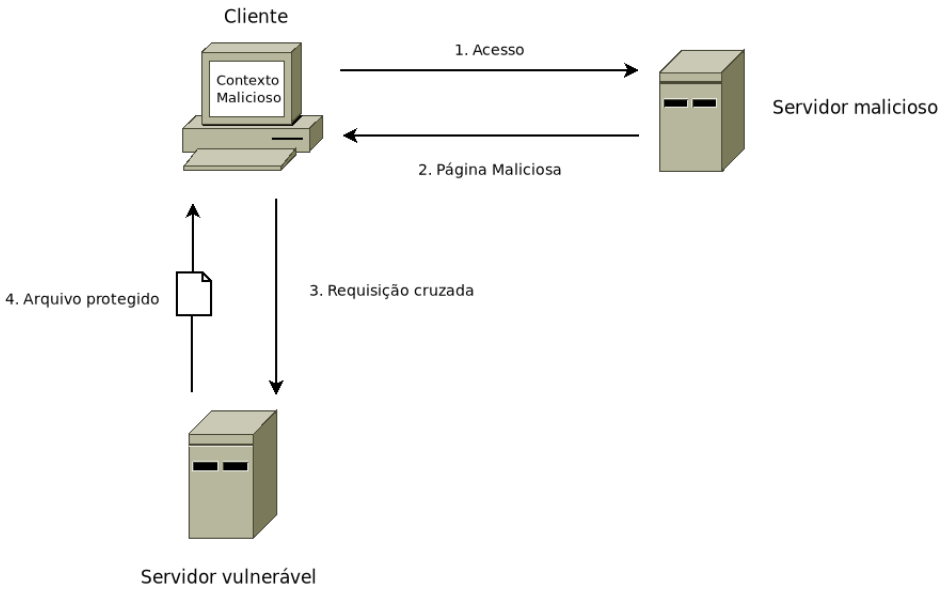

Figura 2. Modelo de interação e exploração da *Cross-Site Script Inclusion*

O comportamento esperado, quando da ocorrência da vulnerabilidade, é o seguinte: o navegador na máquina cliente realiza uma requisição *GET* (1. Acesso) para o servidor malicioso. Este devolve para o cliente uma página maliciosa (2. Página Maliciosa), levando o navegador a realizar uma requisição cruzada (3. Requisição cruzada) de maneira implícita para a aplicação vulnerável, anexando o *cookie*. Isso leva a aplicação vulnerável a permitir acesso e inclusão de um arquivo protegido (4. Arquivo protegido), consequentemente ocorrendo a *Cross-Site Script Inclusion*.

Neste trabalho foram realizados testes dos seguintes casos: *cookies* gerados como *SameSite=None*, *SameSite=Lax* e sem a declaração explícita do atributo *SameSite*. Os

resultados observados para os *cookies* gerados a partir do valor *Lax* tornaram desnecessário testes para *cookies* gerados como *SameSite=Strict*, uma vez que esta configura uma política ainda menos permissiva que a *Lax*.

Além da verificação de diversos navegadores em versões recentes, também foi avaliado o navegador Firefox em suas diversas versões lançadas ao longo dos anos. A finalidade foi buscar por mudanças de comportamento a respeito da vulnerabilidade, que possam ter ocorrido e impactado nos cenários de exploração. A escolha de uma análise aprofundada deste navegador teve por hipótese observar a discrepância de comportamento entre suas versões mais recentes e suas versões lançadas próximas de 2015, sendo interessante uma análise mais detalhada das versões lançadas ao longo do tempo.

4. Descrição dos casos implementados

A aplicação vulnerável foi implementada de maneira a conter arquivos propositalmente suscetíveis à *Cross-Site Script Inclusion*, sendo implementado na aplicação maliciosa as páginas correspondentes para exploração de cada arquivo individualmente, conforme descrito na seção anterior.

O primeiro caso implementado é um aquivo JSONP (JSON *with padding*), que consiste na declaração de um JSON com a adição da declaração de um método para envio destes dados. Essa é uma técnica comumente utilizada por desenvolvedores como uma maneira de burlar as restrições impostas pela política de mesma origem, uma vez que a declaração da função permite a recuperação dos dados a partir do marcador "<*script*>"do HTML. Essa é uma exceção prevista para a política de mesma origem [LEKIES et al. 2021]. A exploração deste arquivo consiste na sua inclusão no contexto da página maliciosa como um *script* e, a partir da função declarada, os dados são recuperados e exibidos em tela. A exploração foi realizada a partir do seguinte código HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JSONP</title>
</head>
<body>
  <h1>Dados do Arquivo JSONP</h1>
  <script>
    function processarDados(dados) {
      console.log('Dados recebidos:', dados);
      document.body.innerHTML += `
        <p><strong>Nome:</strong> ${dados.nome}</p>
        <p><strong>Idade:</strong> ${dados.idade}</p>
        <p><strong>Senha:</strong> ${dados.senha}</p>
      `;
    }
  </script>
  <script src="https://example.duckdns.org/example.jsonp"></script>
</body>
</html>
```

Um arquivo de código Javascript foi o segundo caso implementado, consistindo em um arquivo com a declaração de variáveis e valores para cada uma delas. Este caso é um arquivo de código, podendo também ser incluído a partir do marcador "<*script*>" do HTML, conforme previsto pela política de mesma origem [SULLIVAN and LIU 2011].

A exploração segue modelo similar ao anterior, abusando da utilização dessa *tag*, porém os dados incluídos foram exibidos apenas a partir do console do navegador, conforme o seguinte código HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript</title>
</head>
<body>
<script src="https://example.duckdns.org/example.js"></script>
<script>
  console.log("Nome:", nome);
  console.log("Idade:", idade);
  console.log("Senha:", senha);
</script>
</body>
</html>
```

O último caso implementado consiste em um arquivo de imagem no formato *.jpg*. Arquivos de imagens e de outros formatos de mídia também são exceções previstas pela política de mesma origem [PETTY and THOMPSON 2017], frequentemente consistindo de dados sensíveis para a aplicação a qual as armazena. A exploração deste caso consistiu na sua inclusão a partir do marcador "*<img>*" e exibição em tela, sendo explorada a partir do seguinte código HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Imagem</title>
</head>
<body>

<img src="https://example.duckdns.org/example.jpg">
</body>
```

5. Resultados e Discussão

A primeira avaliação fixou o navegador Mozilla Firefox para testar diferentes versões lançadas ao longo do tempo, em busca de modificações de comportamento para o cenário de exploração da *Cross-Site Script Inclusion*. O *cookie* gerado recebeu o atributo *SameSite=None*.

Observou-se mudança de comportamento a partir da versão 102.1esr, na qual a ocorrência da vulnerabilidade não foi mais possível, ainda que a implementação da *SameSite* tenha ocorrido a partir de uma política permissiva. A partir da observação desse fato, realizou-se uma análise do comportamento do navegador em relação às requisições. O seguinte comportamento foi observado: a tentativa de inclusão foi feita, porém sem a anexação do *cookie*. Como resultado, obteve-se uma resposta com o código de status "403", o que consequentemente não permitiu a exploração da vulnerabilidade de *Cross-Site Script Inclusion* (XSSi). Isso pode ser observado na Figura 3, que contém os cabeçalhos da requisição.

A versão 102.0esr, sendo anterior à mudança de comportamento, realiza a requisição cruzada para a inclusão, porém o *cookie* é enviado como cabeçalho, recebendo

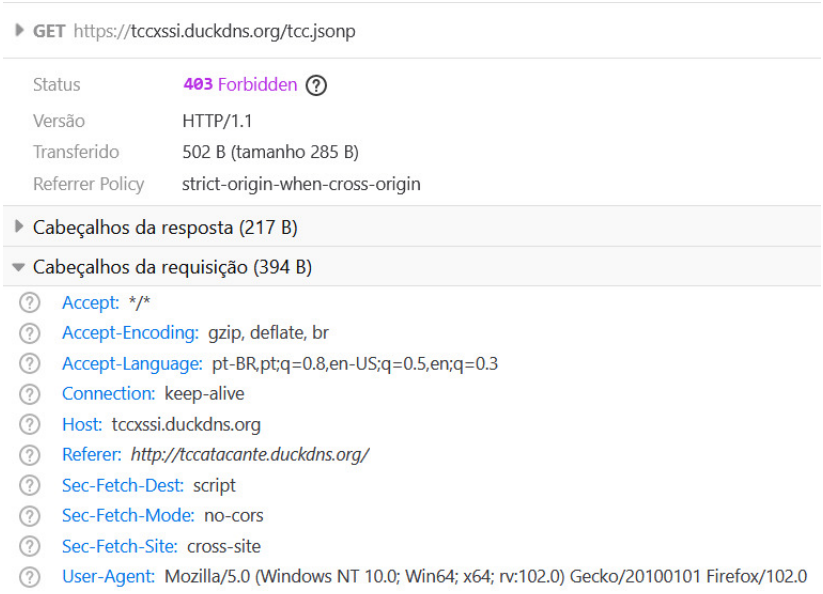

Figura 3. Cabeçalhos da requisição cruzada no *Firefox 102.1esr*

um código ”200” e consequentemente permitindo a exploração da *Cross-Site Script Inclusion*, conforme se verifica nos cabeçalhos contidos na Figura 4. As notas de lançamento da versão 102.1esr não mencionam modificações para a política de mesma origem ou para a implementação da *SameSite*, não sendo claro qual mudança levou a esse novo comportamento.

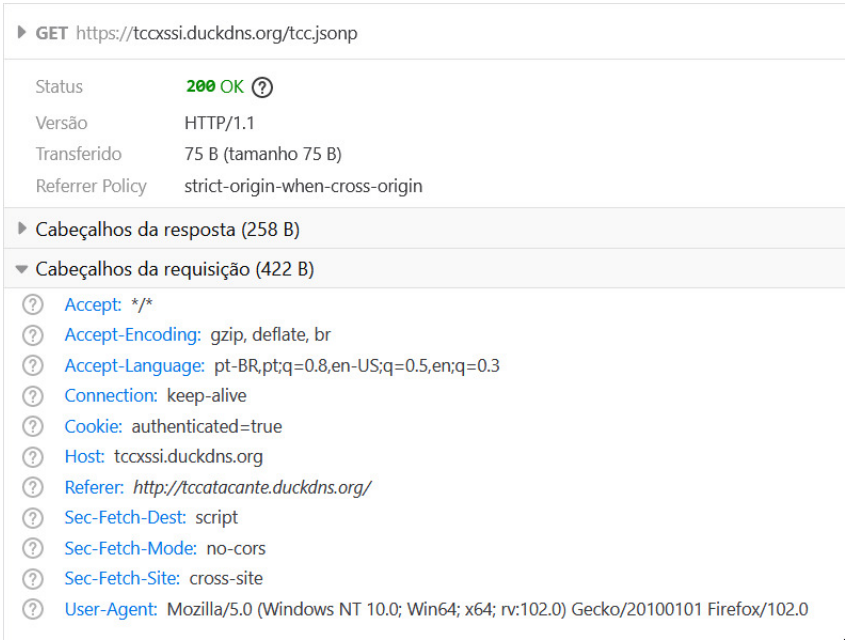

Figura 4. Cabeçalhos da requisição cruzada no *Firefox 102.0esr*

A análise de outros navegadores, em versão atualizada e utilizando o atributo *SameSite=None*, pode ter seu resultado dividido em dois grupos: aqueles navegadores em que a vulnerabilidade ocorreu e aqueles em que a vulnerabilidade não ocorreu, conforme

dados expostos na Tabela 3. Para aqueles aos quais a vulnerabilidade esteve presente, todos os casos implementados foram explorados, enquanto que para aqueles aos quais a vulnerabilidade não esteve presente, nenhum caso implementado foi explorado.

Um outro dado considerado para a construção da Tabela 3 foi o navegador base utilizado para a construção do navegador testado. No cenário *Desktop*, é comum a utilização de navegadores base, sendo também comum que muitas implementações relacionadas à segurança acabem por ser herdadas.

Tabela 3. Resultado da análise dos navegadores

Navegador	**Versão**	**Ocorreu exploração?**	**Base**	**Lançamento**
Chromium	120.0.6099.224	Sim	Própria	12/01/2024
Chrome	121.0.6167.86	Sim	Chromium	25/01/2024
OperaGX	106.0.4998.61	Sim	Chromium	25/01/2024
Opera	106.0.4998.52	Sim	Chromium	18/01/2024
Edge	120.0.2210.144	Sim	Chromium	17/01/2024
Vivaldi	6.5.3206.57	Sim	Chromium	24/01/2024
Avast Secure Browser	120.0.23647.224	Sim	Chromium	19/01/2024
Firefox	122.0	Não	Própria	23/01/2024
Brave	1.61.120	Não	Chromium	17/01/2024
Tor	13.0.9	Não	Firefox	22/01/2024
LibreWolf	122.0-1	Não	Firefox	24/01/2024

Nos casos em que foi possível a exploração da *Cross-Site Script Inclusion*, o comportamento foi similar ao observado para a versão 102.0esr do Mozilla Firefox: a requisição cruzada de inclusão foi realizada pelo navegador, com a anexação do *cookie*. Para aqueles casos em que não foi possível a exploração, o comportamento observado foi o seguinte: a requisição cruzada de inclusão foi realizada pelo navegador (3. Requisição cruzada), porém sem a anexação do *cookie*, similar ao observado na versão 102.1esr do Mozilla Firefox. A Figura 5 ilustra o modelo de interação ocorrido nos casos em que não houve exploração da *Cross-Site Script Inclusion*.

Para os testes conduzidos utilizando o atributo *SameSite=Lax* ou sem a declaração explícita de valor para o atributo, não houve exploração da vulnerabilidade por nenhum dos navegadores testados para nenhum dos casos implementados, o que torna desnecessário testes com *cookies* gerados a partir da *SameSite=Strict*. A Tabela 4 resume os resultados obtidos através do estudo do valor da *SameSite*.

Tabela 4. Resultado da análise dos valores da *SameSite*

	None	***Lax***	**Sem declaração**
Casos explorados	3	0	0
Navegadores explorados	7	0	0
Navegadores não explorados	4	11	11

Com a análise dos resultados, é possível melhor compreender os cenários em que a

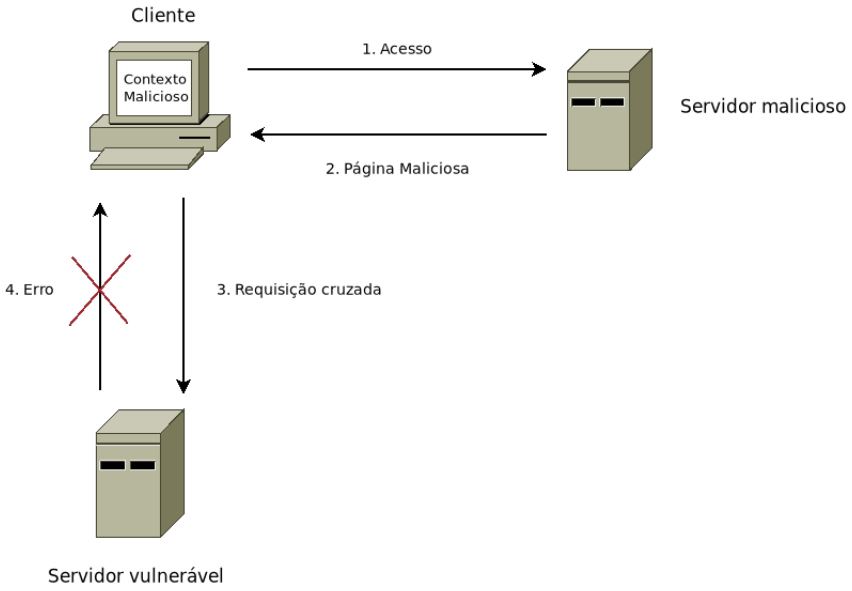

Figura 5. Modelo de interação sem exploração da *Cross-Site Script Inclusion*

vulnerabilidade pode ainda estar presente, bem como as estratégias para mitigação do problema. A título de exemplificação, cita-se uma aplicação mensageira fictícia: um serviço web para troca de mensagens. Tendo uma proposta de maior interatividade em tempo real, a aplicação fictícia utiliza de código Javascript dinâmico constantemente atualizado, a partir de requisições de origem cruzada.

O código dinâmico utilizado pela aplicação fictícia, devido ao alto fluxo trafegado pela aplicação, é recuperado através de uma API armazenada em um domínio externo, sendo necessária a identificação do usuário para a correta recuperação das informações. Para o controle de sessão por parte desta aplicação, são utilizados *cookies* para identificação do usuário, sendo estes também utilizados pela API. Pela necessidade de utilização de requisições de origem cruzada para comunicação com a API, a equipe de desenvolvimento da aplicação utiliza o atributo *SameSite* com o valor *None*, para garantia de que os dados dinâmicos poderão ser enviados entre os domínios, permitindo o funcionamento da recuperação de código dinâmico.

Neste caso, um modelo de ataque viável para exploração da *Cross-Site Script Inclusion*, pode consistir em um agente malicioso desenvolvendo uma página para exploração de arquivos dinâmicos e, através de engenharia social, disseminar entre os usuários da aplicação, levando ao acesso por parte dos mesmos. Com isso, a inclusão de dados potencialmente sensíveis é possível, podendo ser manipulados a partir de uma origem maliciosa.

A exploração da vulnerabilidade poderia ter sido evitada. Para este caso, uma estratégia adequada poderia consistir na utilização do atributo *SameSite* com um valor restritivo, fazendo com que a comunicação entre a aplicação e API seja realizada por meio de outros mecanismos. Um exemplo de mecanismo é o CORS "*Cross-Origin Resource Sharing*", podendo ser declarado como uma lista de exceção, permitindo a comunicação entre os domínios desejados, porém sem expandir a permissão para qualquer outro domínio [ALCORN et al. 2014].

Outras estratégias de mitigação não baseadas na *SameSite* também são possíveis, como a geração de caminhos dinâmicos ou checagem de origem da requisição cruzada. Essas estratégias não baseadas na *SameSite* devem estar aplicadas em todos os arquivos passíveis de exploração, gerando múltiplos pontos possíveis de falha, sendo uma estratégia arriscada.

6. Conclusões e Trabalhos Futuros

A partir dos resultados levantados na seção anterior, entende-se a importância de uma configuração adequada do atributo *SameSite* para mitigação da *Cross-Site Script Inclusion*. A implementação desse atributo pelos navegadores apresentou um grande salto nas estratégias de mitigação para o problema, oferecendo uma alternativa centralizada para solução.

Ao se referir a essa estratégia como ”centralizada”, o sentido atribuído é o de que a implementação da solução ocorre em apenas um item da aplicação: os *cookies*. Neste caso, as equipes de desenvolvimento e de segurança devem se certificar de que os *cookies* de sessão, aqueles aos quais permissões são atreladas, estão implementados a partir de uma política não permissiva, concentrado apenas nesta função o único ponto de falha. Caso necessário a utilização desses *cookies* em requisições de origem cruzada, é crucial a implementação de uma política *SameSite* pouco permissiva, porém aliada de mecanismos como, por exemplo, o CORS ”*Cross-Origin Resource Sharing*”. Isso permite a declaração de domínios incluídos como exceção de maneira explícita, a partir de uma lista de permissão, evitando a universalidade da exceção e consequente inclusão por parte de qualquer domínio.

Outras estratégias avaliadas por outros pesquisadores, anteriormente à implementação do atributo *SameSite* pelos navegadores, como a utilização de caminhos dinâmicos e separação do código estático de dados sensíveis, por exemplo, apresentam múltiplos pontos de falha. Neste caso, deve-se atentar para que todas as partes da aplicação que sejam compostas por recursos passíveis de exploração, estejam protegidas a partir dessas estratégias. Considerando aplicações com múltiplos pontos e em constante expansão, a proteção contra *Cross-Site Script Inclusion* se torna uma tarefa muito mais difícil e passível de falhas, em comparação com uma estratégia baseada nos *cookies* de sessão.

A partir disso, conclui-se que, com as atuais ferramentas, a estratégia de mitigação para a *Cross-Site Script Inclusion* deve estar centrada no atributo *SameSite* e sua adequada configuração, evitando uma política permissiva e utilizando de outros mecanismos quando da necessidade de realização de requisições cruzadas utilizando *cookies*. Porém, ainda assim, deve-se atentar para a importância da separação do código estático dos dados sensíveis, uma vez que esta ainda configura uma má prática de segurança e que pode levar à ocorrência de outros problemas. Em adição, afirma-se da necessidade de atualização de diversos materiais de consulta disponíveis na internet a respeito do tema, muitos não englobando as mudanças nos mecanismos referentes a *Cross-Site Script Inclusion*, apresentando estratégias de mitigação ainda desatualizadas, podendo induzir ao erro o indivíduo que as consulta.

Com os dados levantados a partir deste trabalho, conclui-se também de uma divergência de padronização da implementação da política de mesma origem e do atributo

SameSite no que tange às requisições de origem cruzada e anexação de *cookies*. Nota-se uma implementação de maior e menor permissibilidade quando da utilização da *SameSite* com o valor *None*, permitindo a ocorrência da *Cross-Site Script Inclusion* em alguns navegadores, enquanto em outros a vulnerabilidade não ocorre. Em termos práticos, conclui-se que o lado cliente hoje possui uma autonomia na mitigação da vulnerabilidade, descentralizando apenas do lado servidor, deixando ao usuário a possibilidade de evitar a ocorrência do problema, ainda que a aplicação esteja vulnerável.

6.1. Trabalhos Futuros

A partir deste trabalho, foi identificada mudança no comportamento do navegador Firefox a partir da versão 102.1esr, porém não sendo claro a maneira como a modificação foi implementada e se existem outras implicações além da identificada. Dado o exposto, atribui-se como uma possibilidade de trabalho futuro a análise do código fonte das versões 102.0esr e 102.1esr, em busca da modificação implementada e das mudanças na lógica do navegador. Estendendo este trabalho futuro, pode-se avaliar as divergências de implementação da política de mesma origem e *SameSite* nos diferentes navegadores no nível do código fonte, uma vez que alguns dos navegadores aqui avaliados, possuem seu código fonte público, permitindo portanto essa análise.

A análise de navegadores realizada neste trabalho levou em conta apenas o ambiente *desktop*, porém outros ambientes também contam com implementações de navegadores e são amplamente utilizado por usuários. Uma outra possibilidade de trabalho futuro é a análise do cenário de exploração da *Cross-Site Script Inclusion*, da implementação da *SameSite* e do comportamento dos *cookies* em requisições de origem cruzada nestes outros ambientes. São exemplos de ambientes a serem analisados: *mobile*, *Smart Tvs*, consoles de videogames, dispositivos IoT de maneira geral.

Este trabalho demonstrou que, em um dos cenários, é possível incluir arquivos de mídia como alvo da *Cross-Site Script Inclusion*. Porém, os impactos resultantes desta inclusão devem ser melhor avaliados, uma vez que, apesar de configurar a possibilidade de inclusão e exceção à política de mesma origem, existem restrições quanto à manipulação destes arquivos pelo código Javascript no contexto da aplicação maliciosa. Sugere-se como trabalho futuro um estudo aprofundado das possibilidades de manipulação, tanto de imagens como de outros arquivos de mídia, no contexto da *Cross-Site Script Inclusion*, tendo por finalidade a avaliação do impacto do problema em cenários reais.

Também foi demonstrado que alterações em mecanismos de segurança nos navegadores representaram impactos e mudanças de paradigmas ao se tratar de vulnerabilidades já estudadas e conhecidas. Como trabalho futuro, pode-se conduzir um estudo para avaliação de como essas mudanças podem ter impactado outras vulnerabilidades baseadas em requisições de origem cruzada, bem como da adequação das referências atuais a respeito dessas mudanças.

A implementação de novos mecanismos de segurança em navegadores podem estar atreladas ao surgimento de novos problemas de segurança. Uma possibilidade de trabalho futuro pode ser o estudo dos diferentes mecanismos de segurança para requisições de origem cruzada que foram implementados ao longo dos anos. A partir disso, pode-se conduzir um estudo a respeito dos possíveis erros de implementação por parte de desenvolvedores, seus possíveis impactos e soluções.

Referências

ALCORN, W., FRICHOT, C., and ORRÚ, M. (2014). *The Browser Hacker's Handbook*. John Wiley Sons, Inc.

FRANKEN, G., VAN GOETHEM, T., and JOOSEN, W. (2018). *Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies*. 27th USENIX Security Symposium.

GROSSMAN, J. (2006). Advanced web attack techniques using gmail. https://blog.jeremiahgrossman.com/2006/01/advanced-web-attack-techniques-using.html. Accesso em: Mai. 2024.

HAILPERIN, V. and RUEF, M. (2016). Cross-site script inclusion a fameless but widespread web vulnerability class. https://www.scip.ch/en/?labs.20160414. Accesso em: Mai. 2024.

KERN, C., DASWANI, N., and KESAVAN, A. (2007). *Foundations of Security: What Every Programmer Needs to Know*. Apress.

KHODAYARI, S. and PELLEGRINO, G. (2022). *The State of the SameSite: Studying the Usage, Effectiveness, and Adequacy of SameSite Cookies*. IEEE.

Kurose, J. F. and Ross, K. W. (2009). *Redes de Computadores e a Internet: uma abordagem top-down*. 5ª edição. São Paulo, SP: Pearson Addison Wesley.

LEKIES, S., ENGELS, D., and MITKOV, M. (2021). *JSONPS: Secure an inherently insecure practice with this one weird trick!* 2021 IEEE European Symposium on Security and Privacy Workshops (EuroSPW).

LEKIES, S., STOCK, B., WENTZEL, M., and JOHNS, M. (2015). *The Unexpected Dangers of Dynamic JavaScript*. 24th USENIX Security Symposium.

Mozilla (2023a). Cookies HTTP. https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Cookies. Accesso em: Mai. 2024.

Mozilla (2023b). Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. Accesso em: Mai. 2024.

PETTY, D. and THOMPSON, J. (2017). *The Not-So-Same-Origin Policy*. Independent Security Evaluators Whitepaper.

SULLIVAN, B. and LIU, V. (2011). *Web Application Security: A Beginner's Guide*. McGraw Hill.

TERADA, T. and BUSSAN, M. (2015). *Identifier based XSSI attacks*. MBSD Technical Whitepaper.

WEST, M. and GOODWIN, M. (2016). Internet Draft Same-site Cookies. https://datatracker.ietf.org/doc/html/draft-west-first-party-cookies-07. Accesso em: Mai. 2024.