HENRIQUE CURI DE MIRANDA

INCLUSÕES DE ORIGEM CRUZADA: UM ESTUDO DA VULNERABILIDADE CROSS-SITE SCRIPT INCLUSION, CENÁRIOS E PREVALÊNCIA NOS NAVEGADORES

LAVRAS – MG

2024

HENRIQUE CURI DE MIRANDA

INCLUSÕES DE ORIGEM CRUZADA:
UM ESTUDO DA VULNERABILIDADE CROSS-SITE SCRIPT INCLUSION, CENÁRIOS E PREVALÊNCIA NOS NAVEGADORES

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Graduação em Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Prof. Dr. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG

2024

HENRIQUE CURI DE MIRANDA

INCLUSÕES DE ORIGEM CRUZADA: UM ESTUDO DA VULNERABILIDADE CROSS-SITE SCRIPT INCLUSION, CENÁRIOS E PREVALÊNCIA NOS NAVEGADORES

CROSS-SITE INCLUSION: A STUDY OF THE CROSS-SITE SCRIPT INCLUSION VULNERABILITY, SCENARIOS AND PREVALENCE IN BROWSERS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Graduação em Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 12 de Junho de 2024.

Prof. Dr. Rafael Serapilha Durelli	UFLA
Me. Renan Villela Oliveira	UFLA

gov.br
Documento assinado digitalmente
LUIZ HENRIQUE ANDRADE CORREIA
Data: 30/06/2024 21:20:43-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2024

Em memória dos meus avós: Elias Curi, Maria Gonçalves Pereira e Paulo Rodrigues de Miranda.

AGRADECIMENTOS

Agradeço ao meu professor e orientador Luiz Henrique, ao meu ex-colega de trabalho Jodson Santos e a Nikolaos Eftaxiopoulos, minha liderança no trabalho. Agradeço pelo conhecimento transmitido e pelo auxílio e inspiração para este TCC.

RESUMO

A *Cross-Site Script Inclusion* (*XSSi*) consiste em uma classe de vulnerabilidades que abusa de requisições de origem cruzada e anexação de cookies nas mesmas, sendo fonte de forte negligência ao longo dos anos, tanto por pesquisadores quanto por desenvolvedores e mantenedores de aplicações web. Este trabalho analisa a vulnerabilidade a partir das mudanças ocorridas em navegadores e políticas de segurança nos últimos anos. Foram analisados três cenários de inclusão a partir de um ambiente simulado, sendo observado o comportamento de diversos navegadores em suas versões mais recentes e um navegador específico em diferentes versões, além de analisar o impacto da implementação do atributo *SameSite* aos cookies. Foi observado mudanças no comportamento dos navegadores atuais, alguns permitindo e outros não permitindo a ocorrência da vulnerabilidade. A mudança de comportamento observada no navegador específico e nos navegadores atuais estão relacionadas com a inclusão ou não dos cookies em requisições de origem cruzada, tendo o atributo *SameSite* representado impacto. As estratégias de mitigação para a vulnerabilidade podem ser atualizadas com recomendação de configuração adequada dos cookies de sessão. Além dos cenários clássicos analisados, a inclusão de arquivos de mídia podem representar impacto para a segurança cibernética.

Palavras-chave: Segurança cibernética. Cookies. Política de Mesma Origem.

ABSTRACT

Cross-Site Script Inclusion (XSSI) is a class of vulnerabilities that exploits cross-origin requests and cookie attachment within them, having been a source of significant oversight over the years, both by researchers and by web application developers and maintainers. This work analyzes the vulnerability based on changes that have occurred in browsers and security policies in recent years. Three inclusion scenarios were analyzed using a simulated environment, observing the behavior of various browsers in their latest versions and a specific browser in different versions. The impact of implementing the SameSite attribute on cookies was also examined. Changes in the behavior of current browsers were observed, with some allowing and others not allowing the vulnerability to occur. The observed change in behavior in the specific browser and current browsers is related to whether cookies are included in cross-origin requests, with the SameSite attribute having an impact. Mitigation strategies for the vulnerability can be updated with recommendations for proper session cookie configuration. In addition to classic scenarios analyzed, the inclusion of media files can also have an impact on cybersecurity.

Keywords: Cybersecurity. Cookies. Same-Origin Policy.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O seguinte capítulo tem por objetivo apresentar uma contextualização geral a respeito da vulnerabilidade *Cross-Site Script Inclusion*, bem como a motivação e objetivos deste trabalho. Também é apresentado a maneira como os capítulos foram organizados ao longo do documento.

1.1 Contextualização

Ao se gerir uma aplicação web, um dos fatores de preocupação é a segurança dos recursos ali armazenados. Frequentemente faz-se necessário o armazenamento de dados sensíveis e/ou privados pertencentes ou não aos usuários. Para gerenciamento do acesso a esses dados, *cookies* são comumente utilizados como forma de identificar o usuário, suas permissões e até para geração dinâmica de arquivos (STUTTARD; PINTO, 2011).

No ano de 2006, Jeremiah Grossman descreveu em seu blog pessoal um modelo de ataque capaz de explorar uma vulnerabilidade no Gmail, sendo possível o acesso não autorizado aos contatos de um usuário. Apesar de breve, Grossman descreveu pela primeira vez uma vulnerabilidade que, posteriormente, recebeu o nome de *Cross-Site Script Inclusion* (XSSi), ainda que não tenha sido batizada por Grossman (GROSSMAN, 2006).

Ao longo dos anos, muitos outros pesquisadores se dispuseram a estudar o tema e trabalhos de forte relevância foram publicados. Com isso, o modelo de ataque inicialmente proposto se consolidou como uma classe de vulnerabilidades, com diversas formas possíveis de exploração e impactos, baseados no abuso dos *cookies* para acessar recursos.

Quando se trata de segurança da informação, comumente autores se referem aos pilares da segurança da informação ao tratar dos impactos de um problema, autores diversos citam diferentes números e nomes para esses pilares. Um dos modelos mais populares consiste em: Disponibilidade, Integridade e Confi-

dencialidade. Considerando tal modelo, a vulnerabilidade de XSSi viola o pilar da confidencialidade, uma vez que permite ao agente malicioso o acesso a informações privadas ao qual o mesmo não deveria possuir pelo fluxo normal de funcionamento de uma aplicação (NWEKE, 2017).

Outro fato relevante para contextualização do problema é o funcionamento das aplicações web e navegadores. Houve um tempo em que as páginas web eram em sua maioria estáticas, período este não muito fértil para exploração da referida vulnerabilidade. Hoje contamos com um universo web muito mais dinâmico, com a utilização de *cookies* e diversos recursos passíveis de serem alvos de *Cross-Site Script Inclusion* (STUTTARD; PINTO, 2011).

1.2 Motivação

Apesar dos diversos estudos ocorridos ao longo dos anos a respeito do tema, hoje as pesquisas vivem um período de relativo hiato sobre o assunto, muito motivado por uma maior preocupação geral, tanto da comunidade acadêmica quanto de outros profissionais ligados a segurança cibernética, para com outros problemas e vulnerabilidades.

Um dos projetos de maior relevância para a segurança de aplicações web é o OWASP (Open Web Application Security Project), sendo utilizado de referência por diversos profissionais interessados a respeito do assunto. Um dos recursos mantidos pela OWASP é a classificação dos dez maiores riscos para a segurança das aplicações web, sendo atualizado em intervalos de anos. A vulnerabilidade de *Cross-Site Script Inclusion* nunca esteve incluída nessa classificação, contando apenas com uma página no site da OWASP (OWASP, 2020).

Apesar disso, o problema apresenta uma ameaça real à segurança das aplicações, conforme demonstrado por pesquisas anteriores, e presente em diversos cenários reais, podendo ser ativamente explorada por agentes maliciosos.

Dadas as observações anteriores, entende-se a necessidade do estudo da vulnerabilidade e sua aplicação no contexto atual, envolvendo tecnologias recentes para análise da ocorrência, viabilidade e impacto da exploração do problema. Além disso, para combater essa brecha de segurança, fazem-se necessárias técnicas e tecnologias baseadas em informações oriundas de estudos compatíveis com a realidade atual do cenário web.

1.3 Objetivos

Este estudo tem por objetivo geral analisar a vulnerabilidade *Cross-Site Script Inclusion* em um contexto atualizado.

Os objetivos específicos deste trabalho são:

1. Estudar seu comportamento nos navegadores atuais;
2. Analisar cenários baseados na realidade das aplicações web;
3. Verificar o impacto do atributo *SameSite* na *Cross-Site Script Inclusion*.

Com isso, torna-se possível o entendimento do ambiente em que a vulnerabilidade pode ser encontrada e explorada atualmente, seus possíveis impactos e soluções para mitigação do problema.

1.4 Organização do trabalho

O Capítulo 2 apresenta o referencial teórico, consistindo na apresentação do conhecimento base de conceitos necessários para a compreensão do tema deste trabalho, bem como dos trabalhos relacionados anteriormente desenvolvidos por outros pesquisadores.

É apresentado no Capítulo 3 a classificação do trabalho realizado, além dos procedimentos metodológicos, estes consistindo nos serviços e aplicações utilizados, bem como da interação entre eles.

Os resultados obtidos por este estudo e a discussão resultante destes, são encontrados no Capítulo 4.

No Capítulo 5 encontra-se as conclusões tomadas a partir do capítulo anterior, sendo também encontrados os trabalhos futuros possíveis a partir deste.

2 REFERENCIAL TEÓRICO

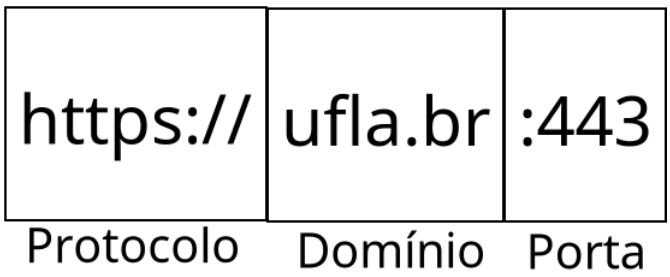
Neste capítulo são apresentados conceitos básicos utilizados para a construção deste trabalho, além de necessários para a compreensão do mesmo. Também são apresentados trabalhos desenvolvidos anteriormente por outros pesquisadores.

2.1 Política de Mesma Origem

A política de mesma origem consiste em um mecanismo crítico para a segurança dos navegadores, onde a mesma é implementada. De maneira geral, tem por objetivo gerar isolamento entre recursos oriundos de origens distintas (MOZILLA, 2023c).

Antes de se definir o conceito de origem, entender a estrutura de uma URL (*Uniform Resource Locator*) é de suma importância. De forma geral, uma vez que outras estruturas podem estar presentes, podemos resumir uma URL em: protocolo, domínio e porta (INWG, 1994). Um exemplo é a Figura 2.1.

Figura 2.1 – Análise da URL



Fonte: O autor

Um dos conceitos importantes a serem estudados é a definição de origem e o que exatamente significa uma mesma origem. Uma origem é definida pelas três estruturas anteriormente citadas: protocolo, domínio e porta. Portanto, quando falamos de uma mesma origem, ambos os três devem ser idênticos. Qualquer

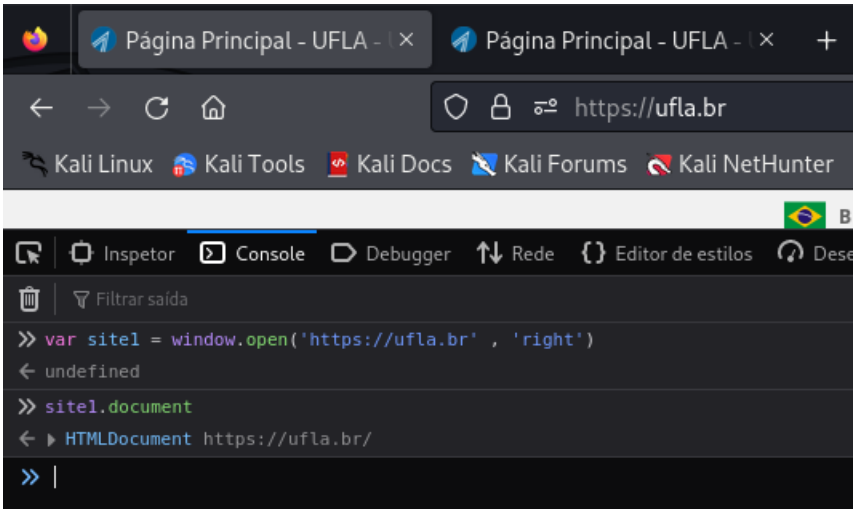
exceção é considerada origem distinta e, por consequência, requisições entre elas serão consideradas de origem cruzada. Por exemplo:

- **https://ufla.br:443** e **https://ufla.br/home:443** são consideradas mesma origem, pois cumpre o requisito dos três itens;
- **https://ufla.br:443** e **http://ufla.br:80** não são considerados mesma origem, pois ainda que o domínio seja o mesmo, o protocolo e a porta não são os mesmos.

A função da política de mesma origem é gerar um ambiente de isolamento baseado nas origens. Sendo assim, evita que tanto o HTML (*HyperText Markup Language*) (INWG, 1995) quanto scripts de uma origem acabe por interagir com o conteúdo de outra origem.

Para entender o isolamento gerado pela política de mesma origem, temos o seguinte exemplo: em primeiro caso, em uma página e utilizando o console do navegador Firefox na versão 115.8.0 (MOZILLA, 2024a), abriu-se utilizando uma variável uma outra página de mesma origem e tentou-se acessar o documento da nova página, conforme Figura 2.2.

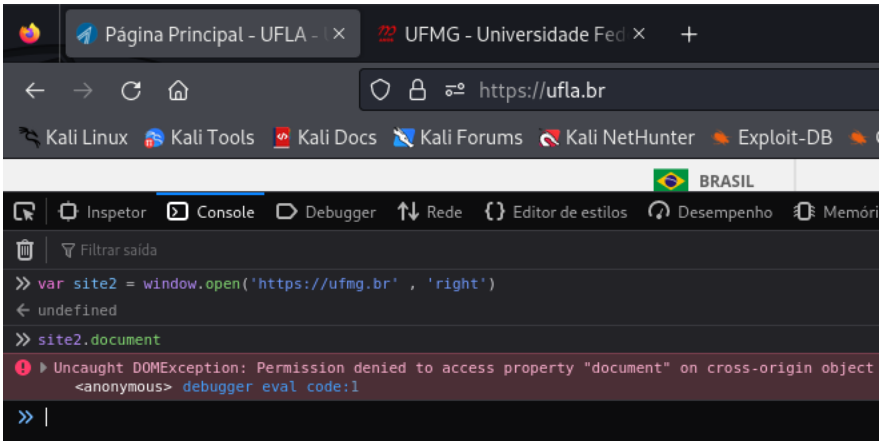
Figura 2.2 – Acesso do documento em uma mesma origem



Fonte: O autor

Conforme se verifica, foi possível acessar o documento, contido no retorno do comando *site1.document*. Ao repetir o mesmo processo para uma página de origem cruzada, o mesmo não se verifica, devido à política de mesma origem, sendo verificado na Figura 2.3.

Figura 2.3 – Acesso do documento em uma origem cruzada



Fonte: O autor

Um outro recurso importante a ser estudado são os *cookies*. De maneira geral, os *cookies* podem ser definidos como cadeias de caracteres armazenados localmente do lado do cliente, utilizados para identificação e armazenamento de informações relativas àquele usuário. Uma das utilizações mais populares dos *cookies* é para a identificação de uma sessão de um usuário, sendo essa sessão relacionada aos privilégios que este determinado usuário possui na aplicação. Portanto, comprometer os *cookies*, frequentemente pode significar comprometer também a sessão como um todo. A origem do cookie se dá devido ao protocolo HTTP (*HyperText Transfer Protocol*) (INWG, 1999), base para as comunicações via navegador, ser considerado *stateless*, ou seja, não guarda estado. Para contornar essa barreira, a implementação dos *cookies* tornou possível a atribuição de cadeias de caracteres para identificar uma sessão, sendo essas atreladas a cada requisição, ge-

rando a sensação de manutenção do estado da comunicação entre cliente e servidor (MOZILLA, 2023a).

No contexto da política de mesma origem, um cookie não pode sofrer interação de uma origem cruzada, como leitura e edição, por exemplo. Porém, em alguns casos, podem ser atrelados a uma requisição de origem cruzada, conforme exceções à política de mesma origem e ao comportamento particular de cada navegador (FRANKEN; GOETHEM; JOOSEN, 2018), conforme estudo desenvolvido neste trabalho e posteriormente relatado.

A política, apesar de crucial para a segurança da web, não é completamente rígida. Visando um melhor compartilhamento de recursos e maior dinamização do conteúdo, existem algumas exceções (MOZILLA, 2023c), por exemplo:

- **Hiperlinks:** essa é uma exceção amplamente utilizada, sendo comum a referenciação e redirecionamento entre sites através de hiperlinks;
- ***Forms:*** através desta exceção, uma origem pode realizar uma requisição do tipo *POST* a outra, sendo a base para ocorrência de uma outra vulnerabilidade: *Cross-Site Request Forgery* (CSRF) (OWASP, 2023);
- **Mídias estáticas:** a inclusão de mídias estáticas, como imagens por exemplo, são uma exceção prevista para a política de mesma origem;
- **Scripts:** a inclusão de scripts externos também é uma exceção, sendo base para exploração da *Cross-Site Script Inclusion* (XSSi). Um fato importante a ser citado nessa exceção é que, em alguns casos, um arquivo no formato JSON (*JavaScript Object Notation*)(IETF, 2017) pode ser tratado como script também (LEKIES; ENGELS; MITKOV, 2021).

Apesar dos diversos usos benignos dessas exceções, como, por exemplo, para métricas de utilização, também foram gerados problemas de segurança e privacidade relacionados, sendo uma delas a vulnerabilidade *Cross-Site Script Inclusion*.

2.2 Atributo *SameSite* dos *cookies*

Conforme citado anteriormente, os *cookies* são utilizados para identificar sessão do usuário. Além disso, os *cookies* podem receber atributos, sendo esses gerados e definidos pelo servidor no momento da criação do *cookie*, porém podendo ser alteradas localmente pelo cliente, ainda que não seja comum. Um desses atributos é o *"SameSite"*, que define a maneira como o navegador deve lidar com o *cookie* em requisições de origem cruzada, sua utilização mais famosa em termos de segurança é a prevenção de ataques de *Cross-Site Request Forgery* (CSRF).

O atributo *SameSite*, tomando como referência o navegador Firefox, passou a ser implementado a partir da versão 60, sendo a mesma lançada no dia 09 de Maio de 2018, hoje suportado por todos os navegadores de maior popularidade. O primeiro navegador popular a implementar o atributo foi o Google Chrome, tendo o mesmo ocorrido na versão 51 de 25 de Maio de 2016. Portanto, a implementação do atributo é um fato relativamente recente na história dos navegadores, apresentando um recurso importante para a segurança das aplicações e para controle das requisições de origem cruzada (MOZILLA, 2024b).

Ao se definir um *cookies* no navegador cliente, o atributo SameSite pode receber os seguintes valores:

- ***Strict:*** é o valor mais restritivo. Neste caso o cookie não será enviado em nenhuma requisição de origem cruzada;

- ***Lax:*** é o valor padrão atribuído quando não existe declaração explícita e um intermediário em termos de restrição. Neste caso o *cookie* não será enviado em todas as requisições de origem cruzada, é enviado quando o usuário navega de um site a outro, ao seguir um link, por exemplo;

- ***None:*** é o valor mais permissivo. Neste caso o *cookie* será enviado em todas as requisições de origem cruzada. Para ser definido, o *cookie* deve possuir

também o atributo *Secure*, obrigando a utilização de conexão criptografada para tráfego.

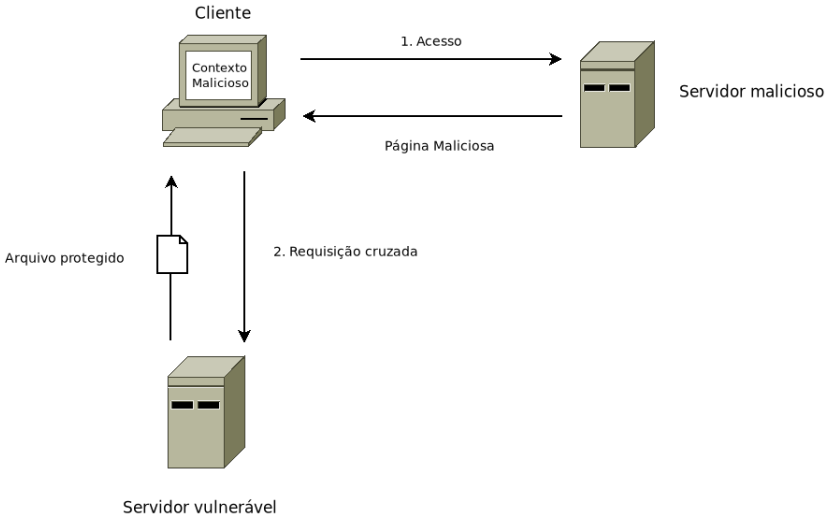
2.3 *Cross-Site Script Inclusion (XSSi)*

Cross-Site Script Inclusion consiste em uma classe de vulnerabilidades que partem de um mesmo princípio. O problema ocorre quando se é possível a inclusão, no contexto de uma página de origem pertencente a um atacante, de recursos inicialmente projetados para não serem acessíveis em outros contextos. Para isso, a vítima deve acessar a página sob domínio controlado pelo atacante, realizando indiretamente também uma requisição de origem cruzada para um domínio legítimo, sendo *cookies* de sessão anexados nesta requisição, para inclusão no contexto da página maliciosa de recursos acessíveis apenas através da sessão do usuário (HAILPERIN; RUEF, 2016).

Um fato importante sobre a vulnerabilidade é que ocorre localmente no navegador do usuário vítima, sendo necessário manipulação dos dados incluídos através do JavaScript e/ou HTML, para se tornar acessível ao atacante (LEKIES et al., 2015).

Cross-Site Script Inclusion é uma vulnerabilidade relativamente similar a outra: *Cross-Site Request Forgery* (CSRF), pois ambas partem do princípio de requisições de origem cruzada que abusam do envio de *cookies* de sessão de um usuário vítima. A diferença ocorre quando, para o CSRF, é comumente feito uma requisição que altera o estado da aplicação legítima alvo, sem a leitura da resposta pelo contexto ou agente malicioso, podendo realizar, a depender do caso específico, alteração de senha, criação de usuário, injeção de dados, etc. Para o caso de XSSi, não é feito alterações na aplicação alvo. O que ocorre é uma inclusão de dados restritos no contexto de uma página maliciosa. O modelo de ataque para XSSi pode ser exemplificado pela Figura 2.4.

Figura 2.4 – Modelo de Ataque da XSSi



Fonte: O autor

Neste exemplo, existe a representação de um cliente que realiza o acesso à uma página maliciosa armazenada em um servidor malicioso, sendo que esta leva o navegador ao envio de uma requisição de origem cruzada para um servidor legítimo com o intuito de incluir recursos, sendo *cookies* anexados nesta requisição. Ao verificar a existência destes *cookies*, o servidor legítimo fornece os recursos solicitados pela página maliciosa, permitindo assim sua inclusão no contexto malicioso.

A engenharia social é outro conceito intrínsecamente ligado à XXSi. Para a ocorrência da vulnerabilidade, é essencial que a possível vítima acesse uma página maliciosa, necessitando ao atacante a utilização de técnicas de persuasão que culminem neste acesso, sendo essas técnicas denominadas como "engenharia social"(CERT, 2012). O agente malicioso pode optar por estratégias generalistas, que possuem como alvo diversas pessoas dentro de um grupo, visando que uma parte delas realizem a ação desejada. Também é possível a opção por técnicas direcionadas, contra um único indivíduo ou contra um pequeno grupo, sendo um

ataque mais específico e que tende a possuir maiores taxas de sucesso, porém atingindo menos pessoas.

2.4 Trabalhos anteriores

2.4.1 Jeremiah Grossman

A primeira publicação a respeito da vulnerabilidade *Cross-Site Script Inclusion* (XSSi) foi feita no blog de Jeremiah Grossman pelo próprio em 2006 (GROSSMAN, 2006). Na ocasião, foi descrita uma falha na aplicação web do Gmail (GOOGLE, 2024), tornando possível o comprometimento da lista de contatos de uma eventual vítima.

A falha de segurança descrita por Grossman inicialmente ocorria ao se enviar para uma possível vítima um link que, ao ser acessado, redirecionaria para uma página ao qual o domínio pertenceria ao atacante. A página possuía uma inclusão de script externo, direcionada a um vetor JSON (*JavaScript Object Notation*) (IETF, 2017) sob domínio do Gmail e preenchido dinamicamente com a lista de contatos do usuário, utilizando os *cookies*, estes atrelados à requisição de origem cruzada. Com isso, foi possível a inclusão deste vetor no contexto da página do atacante que, posteriormente com manipulação no próprio contexto, tornaria possível o acesso dos dados.

2.4.2 Cristoph Kern

Apesar da primeira publicação a respeito da vulnerabilidade ter sido feita por Grossman, o nome *Cross-Site Script Inclusion* somente foi criado a partir do ano seguinte, no livro *"Foundations of Security What Every Programmer Needs to Know"* (KERN; DASWANI; KESAVAN, 2007).

Além disso, a obra formalizou a definição para a vulnerabilidade, dedicando uma subseção para isso, citando o impacto quanto a inclusão de scripts

externos gerados dinamicamente, sendo também incluído um exemplo para ilustração, similar ao caso relatado por Grossman. A obra também inclui uma seção para descrever técnicas de mitigação para a vulnerabilidade.

2.4.3 Nick Nikiforakis

O trabalho desenvolvido por Nikiforakis (NIKIFORAKIS et al., 2012), apesar de não estar diretamente relacionado com a vulnerabilidade XSSi, pode ser considerado interessante para estudo do universo para exploração da mesma. Conforme observado nos trabalhos de pesquisadores anteriores, a exploração da vulnerabilidade se baseia em scripts gerados dinamicamente e passíveis de inclusão.

Nikiforakis conduziu um estudo a respeito da inclusão de código *JavaScript* (MOZILLA, 2022b) de origens externas ao longo dos anos de 2001 a 2010 nos sites Alexa top[1] 10000. Com isso, o trabalho demonstrou uma forte presença de inclusões desta natureza, além de uma tendência de crescimento ao longo dos anos estudados, conforme Tabela 2.1.

Tabela 2.1 – Resultados obtidos por Nikiforakis

Ano	Sem Dados	Mesmas Inclusões	Novas Inclusões	% Novas Inclusões
2001	8,256	1,317	427	24.48%
2002	7,952	1,397	651	31.79%
2003	7,576	1,687	737	30.40%
2004	7,100	2,037	863	29.76%
2005	6,672	2,367	961	28.88%
2006	6,073	2,679	1,248	31.78%
2007	5,074	3,136	1,790	36.34%
2008	3,977	3,491	2,532	42.04%
2009	3,111	3,855	3,034	44.04%
2010	1,920	4,407	3,673	45.46%

Fonte: You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions

[1] Foi um serviço que indexava sites de acordo com o tráfego de visitantes

Com isso, tornou-se possível visualizar o surgimento de um cenário no contexto web ainda mais favorável para exploração da vulnerabilidade.

2.4.4 Takeshi Terada

Nos trabalhos apresentados anteriormente, um ponto em comum é que os dados acessados entre contextos de origens distintas eram tratados como scripts pelo navegador. Até mesmo no trabalho de Grossman, sendo um JSON o alvo, o mesmo era tratado como script devido a natureza particular na qual foi desenvolvido.

Com o trabalho de Takeshi Terada (TERADA, 2015), já em 2015, surgiu o conceito de *"Non-Script-XSSi"*, ao se demonstrar que dados fora de arquivos de script também poderiam ser alvo de *Cross-Site Script Inclusion*. Um exemplo demonstrado por Terada foi a possibilidade de executar ataques contra arquivos CSV (*Comma-Separated Values*) (INWG, 2005).

2.4.5 Sebastian Lekies

Também no ano de 2015, Sebastian Lekies publica um estudo a respeito da vulnerabilidade e do seu contexto (LEKIES et al., 2015). Foi desenvolvido um estudo específico para o caso de *JavaScript* gerados dinamicamente, estes consistindo em: arquivos que, ao serem acessados através dos *cookies* de um usuário, são dinamicamente gerados de acordo com as informações daquele usuário.

Lekies conduziu um estudo empírico no qual analisou o Alexa Top 150 sites aos quais foi possível realização de cadastro. Foi feito um levantamento em busca de arquivos *JavaScript* dinamicamente gerados e, após isso, a tentativa de exploração da *Cross-Site Script Inclusion* contra estes alvos. Os resultados são observados na Tabela 2.2.

Tabela 2.2 – Resultados obtidos por Lekies

	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

Fonte: Usenix - The Unexpected Dangers of Dynamic JavaScript

Outra informação apresentada por Lekies foi o motivo para o qual alguns alvos não foram passíveis de exploração. Para o sucesso do atacante, um fator necessário é a capacidade de prever o caminho para o arquivo alvo na estrutura da aplicação alvo. Portanto, se o caminho também é dinamicamente gerado, o arquivo não pode ser incluído. Um outro motivo apontado pelo estudo de Lekies para a não exploração da vulnerabilidade é a checagem da origem da requisição através do cabeçalho *"Referrer"*, não sendo permitido o acesso caso a origem fosse diferente do domínio da aplicação.

3 METODOLOGIA

Este trabalho consiste em uma pesquisa de natureza aplicada, uma vez que tem por objetivo o desenvolvimento de conhecimento prático em auxílio da compreensão da vulnerabilidade *Cross-Site Script Inclusion* e sua ocorrência em cenários cotidianos. Sua aplicação pode ocorrer em uma vasta gama de áreas relacionadas com a segurança da informação, desde o desenvolvimento seguro de aplicações até para testes de penetração. (GIL, 2017)

Quanto ao objetivo geral, este trabalho se classifica como uma pesquisa descritiva, pois, seu objetivo consiste em observar e registrar a ocorrência e comportamento da vulnerabilidade *Cross-Site Script Inclusion* em ambientes específicos. Dos procedimentos, consiste em um estudo de caso, pois, os procedimentos consistem na investigação da ocorrência de uma situação em cenários reais. (GIL, 2017)

3.1 Procedimentos metodológicos

Para os devidos testes necessários ao cumprimento dos objetivos deste trabalho foi criado um ambiente simulado consistindo em: uma aplicação vulnerável, uma aplicação maliciosa e um cliente vítima.

As aplicações maliciosa e vulnerável foram hospedadas em nuvem utilizando o serviço “AWS Lightsail” (AMAZON, 2024), sendo acessível pela internet e possuindo, portanto, um endereço IP público, atrelado a um domínio através de DNS (*Domain Name System*) dinâmico (INWG, 1987) utilizando o serviço “DuckDNS” (DUCK DNS, 2024). Também foi utilizada uma conexão criptografada através do protocolo HTTPS (*hypertext transfer protocol secure*) (INWG, 2000), com certificado gerado através da ferramenta “Certbot” (EFF, 2024). Para ambas o sistema operacional utilizado foi o “Debian” 6.1.85-1 (DEBIAN, 2022) e o servidor web foi o “Apache” 2.4.59 (APACHE, 2024).

O cliente vítima consiste em uma máquina virtual “Windows 11 Enterprise Evaluation 22H2” (MICROSOFT, 2024) hospedada através do software de virtualização “VirtualBox” (ORACLE, 2024). Os navegadores utilizados tiveram como referência a versão mais recente disponível em janeiro de 2024, sendo utilizadas as configurações padrões tanto durante quanto após a instalação.

A Tabela 3.1 resume os softwares utilizados.

Tabela 3.1 – *Softwares* utilizados

	Aplicação Vulnerável	**Aplicação Maliciosa**	**Cliente Vítima**
Sistema Operacional	*Debian* 6.1.85-1	*Debian* 6.1.85-1	*Windows 11 Enterprise Evaluation 22H2*
Aplicação	*Apache* 2.4.59	*Apache* 2.4.59	Navegadores diversos

Fonte: O autor

A construção do ambiente se iniciou pelo cliente, através do download do software "VirtualBox" a partir do site oficial e da máquina virtual contendo o sistema operacional "Windows", também a partir do site oficial. Com isto, foi exportado a máquina para o "VirtualBox" e se prosseguiu para a instalação dos navegadores, com instaladores fornecidos pelos respectivos desenvolvedores através do endereço oficial de cada.

A construção da aplicação vulnerável e da aplicação maliciosa seguiram passos similares. Primeiro foi feito cadastro no serviço "AWS Lightsail" e criação de duas máquinas virtuais utilizando a imagem de sistema operacional “Debian 6.1.85-1". Foi feito cadastro no DNS “DuckDNS”, seguido da criação de domínio para ambas as aplicações, sendo atrelados aos respectivos endereços IP. Após, foi gerado certificado para cada aplicação através do “Certbot”, permitindo assim tráfego criptografado através do HTTPS. O processo finalizou com a instalação e ativação do servidor web “Apache”, permitindo desenvolvimento e armazenamento das páginas contidas em cada servidor.

3.1.1 Aplicação vulnerável

A aplicação consiste em uma página de autenticação por *login* e senha, sendo "tcc / tcc"as credenciais aceitas. Ao se realizar autenticação, a aplicação gera um *cookie* a ser armazenado no navegador do usuário com o seguinte valor: *authenticated=true*, sendo realizado teste com duas versões diferentes de atributos: *SameSite=None; Secure* e *SameSite=Lax; Secure*. O código fonte desta página é encontrado no Apêndice A. A função *document.cookie* é responsável por gerar o *cookie*, sendo nela realizada as alterações para se testar diferentes valores para a *SameSite*.

Figura 3.1 – Página de autenticação na aplicação vulnerável

Fonte: O autor

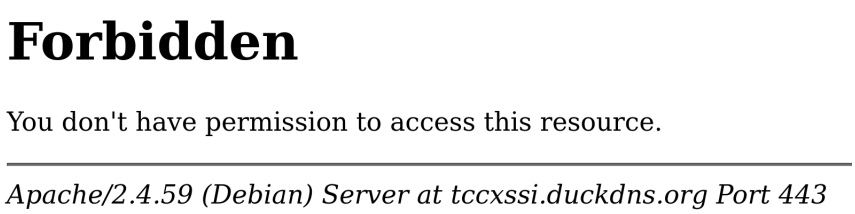
A existência da página de *login* e dos *cookies* tem por objetivo gerar uma área protegida dentro da aplicação, sendo a mesma acessível apenas com a utilização de *cookies* de sessão, tendo as regras devidamente configuradas no arquivo *.htaccess*, encontrado no Apêndice C. Após autenticação, é acessada essa área protegida que consiste em um simples menu com hyperlinks para arquivos vulneráveis, sendo estes também protegidos pelos *cookies* e consistindo em três casos: um JSONP, um código JavaScript e uma imagem, encontrados no Apêndice B.

Figura 3.2 – Menu de arquivos vulneráveis na aplicação vulnerável

Fonte: O autor

O acesso aos arquivos por uma requisição possuindo o *cookie*, retorna um código HTTP 200 *OK* e o acesso é bem sucedido. Caso exista uma tentativa de acesso sem a anexação de *cookie*, o código retornado é o 403 *Forbidden*.

Figura 3.3 – Erro ao acessar o arquivo sem o *cookie*

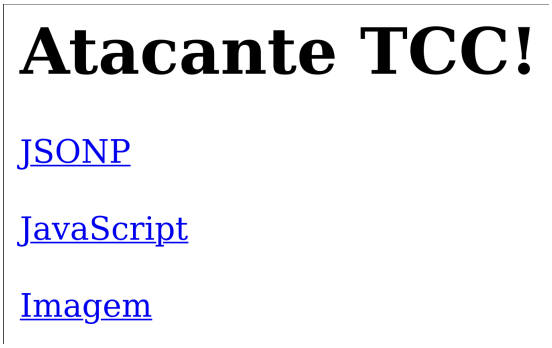


Fonte: O autor

3.1.2 Aplicação maliciosa

A aplicação maliciosa consiste em um menu com hyperlinks para páginas que exploram cada um dos arquivos vulneráveis implementados na aplicação vulnerável, páginas estas com o código fonte contido no Apêndice D.

Figura 3.4 – Menu da aplicação maliciosa



Fonte: O autor

Ocorre para o caso do JSONP um cenário de inclusão do arquivo e exibição dos seus dados na tela. Para o JavaScript é feita a inclusão do código e exibição de suas variáveis através do console. Para a imagem, é feita sua inclusão e exibição da mesma na tela.

3.1.3 Cliente vítima

O cliente consiste na hospedagem de navegadores que acessaram a aplicação maliciosa, possuindo armazenado o *cookie* necessário para o acesso aos arquivos protegidos da aplicação vunerável. Foram realizados dois grupos de testes: o primeiro consiste na avaliação de um mesmo navegador em diferentes versões, em busca de uma mudança de comportamento. E, o segundo, na avaliação de diferentes navegadores em versões atualizadas em busca de comportamentos diferentes. Foram testados os seguintes: Avast Secure Browser, Brave, Chrome, Chromium, Edge, Firefox, Librewolf, Opera, OperaGX, Tor Browser, Vivaldi. Os resultados de cada avaliação são descritos no capítulo seguinte.

4 RESULTADOS E DISCUSSÃO

Neste capítulo serão mostrados os resultados oriundos da metodologia descrita no Capítulo 3. Serão discutidas estratégias de mitigação para a vulnerabilidade *Cross-Site Script Inclusion* e possíveis cenários práticos para a exploração da vulnerabilidade.

A apresentação dos resultados foi dividida em duas: uma primeira seção onde são apresentados os resultados obtidos a partir da análise de um único navegador em diferentes versões, e, uma segunda seção onde pode-se verificar os resultados obtidos com a análise de diversos navegadores em versões únicas e atualizadas.

4.1 Navegador específico

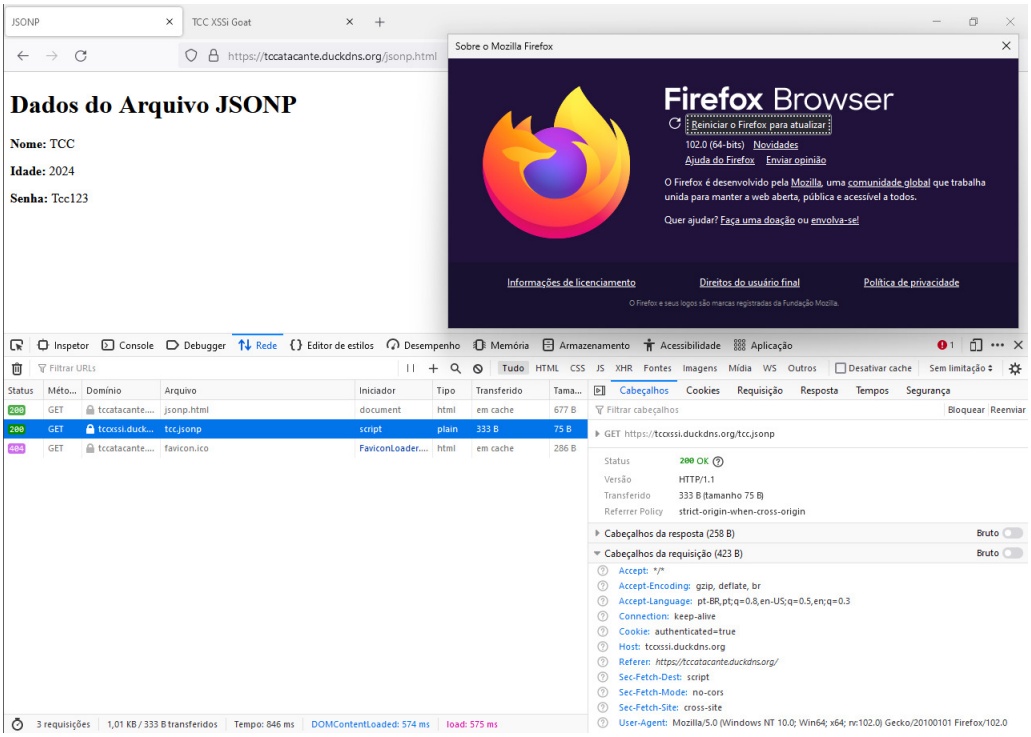
Foram realizados testes práticos em diferentes versões e utilizando o ambiente de testes previamente criado. Notou-se uma mudança de comportamento considerável ao se tomar por referência o navegador Mozilla Firefox. Essa mudança foi percebida a partir da versão 102.1.0esr, lançada em 26 de Julho de 2022, portanto, já compatível com o atributo *SameSite* (MOZILLA, 2022a).

Em versões anteriores à 102.1.0esr, caso o *cookie* tenha sido gerado utilizando os atributos *SameSite=None*, ou, sem declaração explícita do atributo, a vulnerabilidade era presente e ocorria conforme o esperado. Na Figura 4.1, é possível ver a ocorrência da inclusão do arquivo JSONP e a exibição de seus dados.

Ao se verificar a página carregada pelo navegador, percebe-se que os dados contidos no arquivo JSONP, de origem distinta, foram exibidos conforme esperado. Verificando a aba de "Rede"do navegador, é possível visualizar a requisição de origem cruzada realizada para inclusão do mesmo, tendo esta recebida uma resposta de código "200", o que demonstra o sucesso da mesma. Ao verificar os cabeçalhos contidos na requisição, percebe-se também que o *cookie* de sessão utilizado na aplicação vulnerável foi anexado.

Pela aba "Sobre o Mozilla Firefox", é verificado a versão do navegador utilizada, sendo a versão 102.0esr.

Figura 4.1 – Vulnerabilidade ocorrendo na versão 102.0esr com JSONP



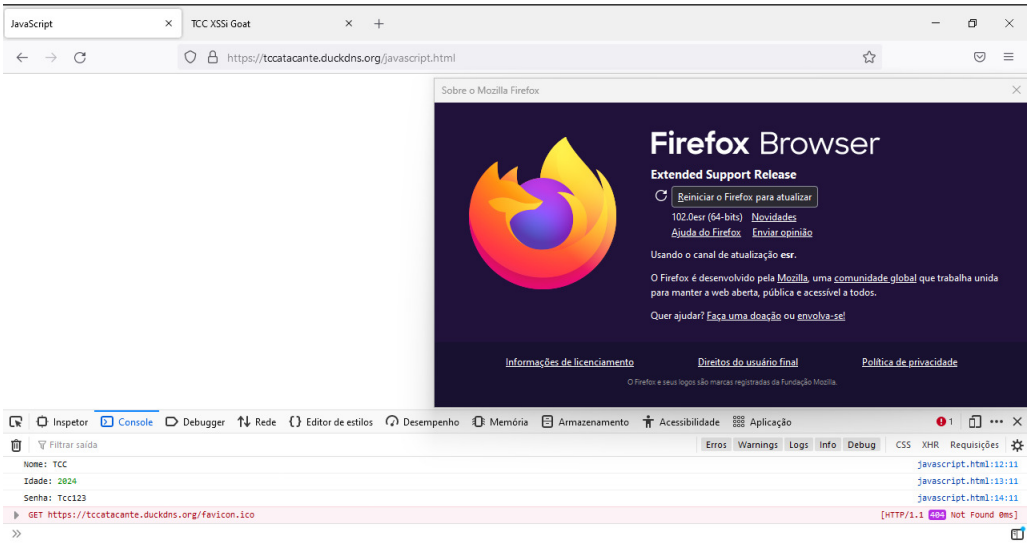
Fonte: O autor

Para o cenário em que existe a inclusão do código JavaScript, o mesmo foi adicionado no contexto da aplicação maliciosa, incluindo suas variáveis. Com essa inclusão, foi possível a manipulação dos dados das variáveis legítimas a partir do contexto malicioso. Em termos práticos, esses dados poderiam ser eviados para um atacante ou manipulados para realização de outras ações maliciosas. No contexto do ambiente simulado, esses dados foram mostrados no console como uma exibição de variáveis, conforme Figura 4.2.

Similar ao exemplo anterior, a versão do navegador é a mesma, conforme aba "Sobre o Mozilla Firefox". Neste caso, assim como o anterior, o comporta-

mento da requisição cruzada foi o mesmo, sendo anexado o *cookie* de sessão e tendo recebido uma resposta de código "200",

Figura 4.2 – Inclusão e exibição das variáveis

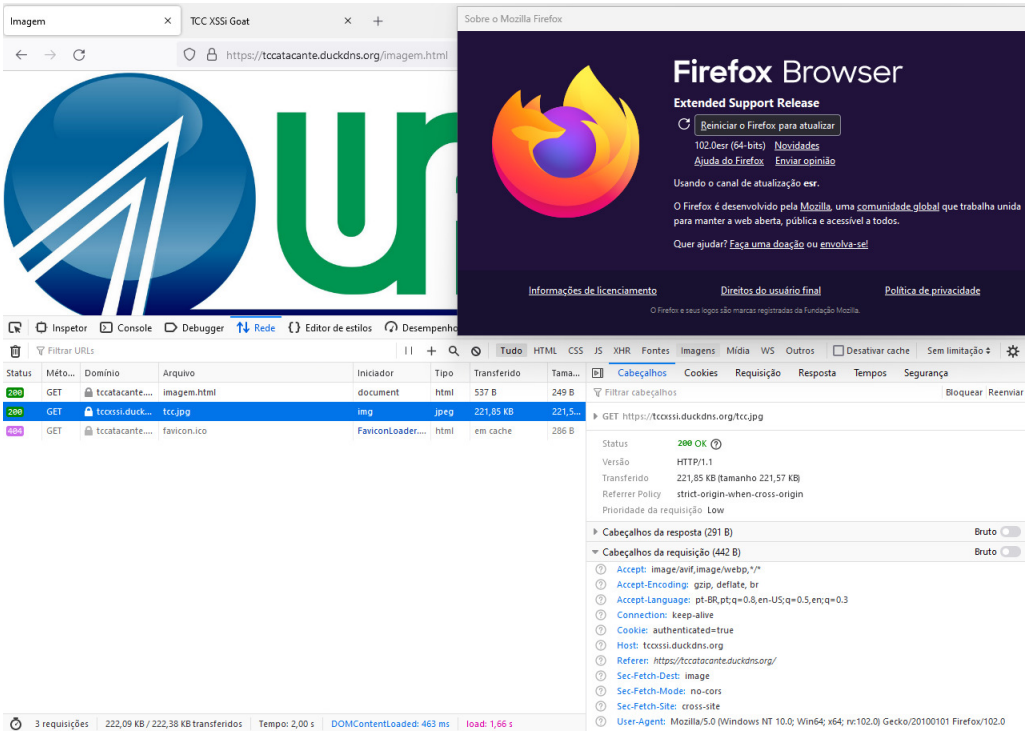


Fonte: O autor

No contexto da imagem, a inclusão também foi possível a partir do abuso da *tag* "<*img*>", sendo a mesma exibida para o usuário a partir da origem maliciosa, conforme Figura 4.3

Similar aos exemplos anteriores, a aba "Rede"permite verificar as requisições realizadas, sendo notada a requisição de origem cruzada responsável pela inclusão da imagem alvo, que recebeu uma resposta de código "200". O *cookie* de sessão também foi anexado para esta requisição, sendo verificado através dos cabeçalhos. A versão do navegador utilizado também se manteve a mesma.

Figura 4.3 – Inclusão e exibição da imagem



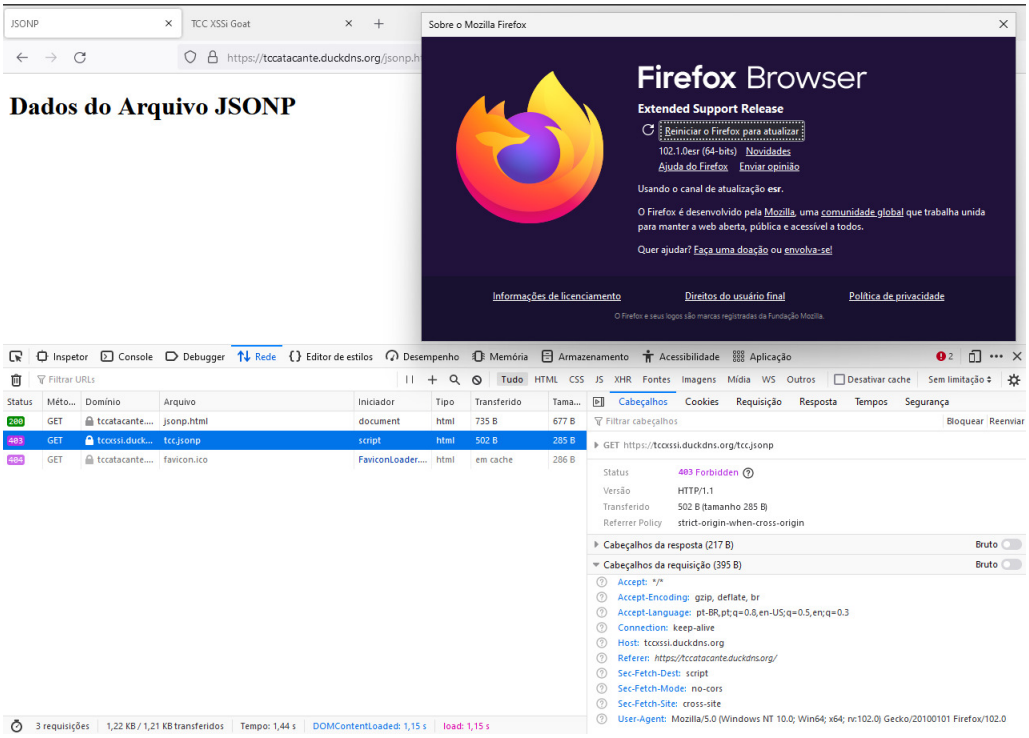
Fonte: O autor

Para outros valores atribuídos ao *SameSite*, não há vulnerabilidade, pois, o *cookie* de sessão não é anexado na requisição cruzada. A ocorrência da vulnerabilidade ao não se declarar explicitamente o atributo é um resultado inesperado, uma vez que, de acordo com as normatizações, a não declaração do atributo define o valor como *Lax*.

Para a versão 102.1.0esr e posteriores, o acesso à página maliciosa ainda leva à ocorrência da requisição de origem cruzada, conforme em versões anteriores e vulneráveis. A diferença de comportamento observada foi quanto à utilização do *cookie* de sessão, sendo este não anexado na requisição cruzada tendo, portanto, recebido a resposta de código 403 *Forbidden* independente do valor do atributo *SameSite*, podendo ser verificado na Figura 4.4.

Diferente do caso da Figura 4.1, neste caso os dados do arquivo JSONP não foram exibidos na página do navegador. O motivo é encontrado na aba "Rede": a requisição cruzada recebeu uma resposta de código "403", evidenciando a falha na inclusão. O que gerou essa falha é observado nos cabeçalhos da requisição, pois, não há anexação do *cookie* de sessão utilizado para controle de acesso ao arquivo na aplicação vulnerável.

Figura 4.4 – Vulnerabilidade não ocorrendo na versão 102.1esr com JSONP



Fonte: O autor

Nas versões em que era possível a vulnerabilidade, todos os três cenários implementados foram explorados com sucesso, enquanto nas versões que não era possível, nenhum cenário implementado foi explorado.

A análise das notas de lançamento da versão 102.1.0esr não citam nenhuma mudança relacionada com os mecanismos do navegador envolvidos na

ocorrência da vulnerabilidade, não sendo claro a mudança na implementação das políticas relacionadas que geraram a alteração de comportamento observada.

4.2 Navegadores analisados

Os navegadores analisados podem ser divididos em dois grupos: os que permitem e os que não permitem a ocorrência da *Cross-Site Script Inclusion*.

Naqueles navegadores que permitem a ocorrência da vulnerabilidade, ela aconteceu ao se definir o *cookie* com o atributo *SameSite=None* apenas, enquanto que da não declaração explícita ou da declaração com um valor diferente, não há ocorrência da vulnerabilidade. Para os navegadores que não permitem a vulnerabilidade, independentemente do valor atribuido ao atributo *SameSite*, não há vulnerabilidade.

A Tabela 4.1 organiza os resultados obtidos nesta etapa.

Tabela 4.1 – Resultado da análise dos navegadores para *SameSite=None*

Navegador	**Versão**	**Permite?**	**Base**
Chromium	120.0.6099.224	Sim	Própria
Chrome	121.0.6167.86	Sim	Chromium
OperaGX	106.0.4998.61	Sim	Chromium
Opera	106.0.4998.52	Sim	Chromium
Edge	120.0.2210.144	Sim	Chromium
Vivaldi	6.5.3206.57	Sim	Chromium
Avast Secure Browser	120.0.23647.224	Sim	Chromium
Firefox	122.0	Não	Própria
Brave	1.61.120	Não	Chromium
Tor	13.0.9	Não	Firefox
LibreWolf	122.0-1	Não	Firefox

Fonte: O autor

Para os navegadores que permitiram a vulnerabilidade, todos os casos foram explorados com sucesso, enquanto que para os que não permitiram, nenhum dos cenários foram explorados.

Dentre o universo de navegadores para Desktop, comumente existe um navegador base utilizado para a contrução, sendo Chromium e Firefox os navegadores base principais. Este foi um dado levado em consideração ao se analisar os resultados obtidos e construir a Tabela 4.1.

4.3 Mitigação da *Cross-Site Script Inclusion*

Ao longo do tempo, diversas estratégias surgiram como métodos utilizados para mitigação da *Cross-Site Script Inclusion*. De maneira geral, uma das mais utilizadas é a segregação entre o código estático e dados dinâmicos, levando uma separação entre o JavaScript e dados pessoais ligados ao usuário da aplicação e consequentemente de sua sessão. A utilização de JSONP para dados sensíveis também é desencorajada, uma vez que é tratada como script pelo navegador e está atrelada a diversos problemas de segurança (HAILPERIN; RUEF, 2016).

Outras estratégias também surgiram, principalmente a partir do trabalho de Sebastian Lekies, demonstrando que a utilização de caminhos dinâmicos e imprevisíveis para arquivos dinâmicos podem ser utilizados, por evitarem a construção de uma página maliciosa que realize a inclusão dos mesmos. Outra estratégia é a verificação do cabeçalho "*Referrer*", que indica a origem da requisição, bloqueando assim qualquer requisição de origem cruzada destinada a um arquivo sensível (LEKIES et al., 2015).

Apesar das estratégias anteriores, o universo dos navegadores e a internet como um todo mudaram ao longos dos anos posteriores a essas pesquisas. Com base nos dados obtidos através deste trabalho, é possível afirmar que uma configuração adequada dos *cookies* de sessão é uma estratégia eficiente para combate ao problema da XSSi. A utilização do atributo *SameSite* com valores diferentes de *None* consistem em uma solução global para o problema, evitando inclusive erros pontuais a partir do servidor.

Em estratégias como, por exemplo, caminho dinâmico ou checagem do do cabeçalho "*Referrer*", deve-se atentar para que a estratégia esteja aplicada em todos os arquivos sensíveis, não sendo portanto uma estratégia global e, por consequência, sendo mais suscetível a erros pontuais. Enquanto isso, ao mitigar o problema a partir de uma estratégia ligada diretamente ao *cookie* que identifica a sessão do usuário, por consequência, todos os recursos acessíveis a partir deste *cookie* também estarão protegidos, sendo a utilização desta a principal estratégia para mitigação da *Cross-Site Script Inclusion*.

Com base nos dados obtidos nesta pesquisa, também é possível afirmar que os usuários podem ter um papel mais ativo para mitigação do problema e proteção dos próprios dados, evitando confiar apenas nos mantenedores dos serviços que utilizam seus dados pessoais.

A utilização de navegadores que não permitem a ocorrência da vulnerabilidade, apesar de erros do lado do servidor, é uma estratégia possível e aplicável pelo usuário, além também da importância de manter o navegador atualizado na versão mais recente disponível, configurando essa estratégia um outro ponto levantado por este trabalho. Além disso, boas práticas gerais podem ser aplicadas: evitar acessar links desconhecidos, principalmente quando ligados a mensagens com senso de urgência ou ofertas muito vantajosas, e evitar a inserção de dados pessoais de maneira excessiva e desnecessária em aplicações pela internet (CERT, 2012).

4.4 Cenários de exploração

A partir da análise dos resultados obtidos e das possibilidades de mitigação para o problema, torna-se possível a construção de cenários fictícios, porém baseados em cenários reais e cotidianos, em que a vulnerabilidade pode estar presente, bem como a análise do impacto e de estratégias práticas para solução eficiente da vulnerabilidade.

4.4.1 Documentos pessoais

A empresa fictícia "TCCQuímicos"realiza a venda de produtos químicos controlados através da internet, apostando em um modelo comum entre lojas, que também realiza a entrega através de uma empresa terceirizada de transportes, sem qualquer interação com o comprador fora do ambiente virtual.

Para estar de acordo com a legislação responsável, a empresa exige de seus compradores o envio prévio de um documento de identificação, consistindo em uma imagem do documento. O arquivo é sempre armazenado em um diretório na aplicação web da loja, seguindo um modelo de nome previsível.

Os desenvolvedores da aplicação acreditam que os documentos estão seguros, pois são acessíveis apenas através dos *cookies* de sessão do usuário correspondente. Porém, para registro de métricas de uso da aplicação por cada usuário, a TCCQuímicos utiliza uma ferramenta externa armazenada em um outro domínio, utilizando uma política de *cookies* permissiva, tendo o atributo "*SameSite=None*", para que a ferramenta colete adequadamente os dados de cada usuário e os processe.

Um agente malicioso, em busca de imagens de documentos de identificação para cometimento de fraudes bancárias, identifica a falha de segurança na aplicação da TCCQuímicos e busca explorar para seu objetivo. Para isso, o agente malicioso acessa uma página de rede social ligada a empresa alvo e verifica as avaliações deixadas anteriormente por compradores, sendo identificável o usuário "Fulano", que avaliou uma compra realizada recentemente na empresa.

Com essas informações e com o conhecimento previsível do caminho para o arquivo que corresponde ao documento de Fulano, o agente malicioso desenvolve uma página web fraudulenta com o intuito de fazer a inclusão da imagem. Após isso, o atacante envia uma mensagem a Fulano se passando por um representante comercial da TCCQuímicos, oferecendo uma oferta muito vantajosa a partir do acesso ao link malicioso. Caso Fulano realize o acesso, o mesmo es-

tará passível de ter seu documento de identificação comprometido a partir de uma vulnerabilidade *Cross-Site Script Inclusion*.

O cenário anterior poderia ter sido evitado por diversos fatores. Pela necessidade de identificação do usuário entre origens distintas, a aplicação poderia realizar a implementação de dois *cookies* distintos, sendo um para identificação do mesmo na aplicação e o gerenciamento de suas respectivas permissões, enquanto o outro seria utilizado exclusivamente para sua identificação para a ferramenta externa sem possuir qualquer grau de permissão na aplicação original. Pelo lado de Fulano, caso o mesmo não tivesse acessado o link malicioso por cautela com a legitimidade do mesmo, ou caso utilizasse um navegador com maior proteção contra *Cross-Site Script Inclusion*, o comprometimento de seu documento poderia ter sido evitado.

4.4.2 Mensageiro

Com a popularização das aplicações mensageiras, surgiu no mercado a aplicação fictícia "TCCMensagens", se propondo como uma alternativa aos mensageiros consagrados, prometendo eficiência em sua aplicação, acessada via internet através de um navegador.

Com o intuito de uma melhor interatividade em tempo real, a aplicação utiliza código JavaScript dinâmico, constantemente atualizado, consistindo nos contatos e nas mensagens mais recentes por eles enviadas ao usuário do mensageiro. O arquivo responsável por essa funcionalidade, ao ser solicitado, a aplicação verifica os *cookies* de sessão anexados na requisição e, com a devida identificação do usuário, preenche o arquivo com as informações correspondentes.

Devido ao alto fluxo de mensagens pela aplicação, a equipe de desenvolvimento utiliza uma API (*Application Programming Interface*) para a comunicação entre o serviço web e o banco de dados contendo as mensagens do usuário, estando a API amazenada em um outro domínio. Para o adequado funcionamento e iden-

tificação do usuário correto pela API, os *cookies* de sessão foram definidos tendo o atributo "*SameSite=None*", permitindo sua anexação em requisições de origem cruzada.

Um agente malicioso, aproveitando da crescente popularidade do TCC-Mensagens, decide testar a segurança da aplicação e descobre a existência da vulnerabilidade *Cross-Site Script Inclusion* a partir do arquivo dinâmico supracitado. Este atacante, tendo o objetivo de realizar ataques contra empresas e indivíduos ligados à segurança cibernética, decide explorar a falha em busca de obtenção de informações privilegiadas.

Para isso, após o desenvolvimento de uma página para exploração da falha e armazenada em um domínio malicioso, o atacante divulga exaustivamente em redes sociais e em meios ligados à segurança cibernética como um todo, seu novo grupo no mensageiro TCCMensagens para compartilhamento gratuito de cursos pirateados.

Ao reunir um grande grupo de pessoas interessadas no assunto e, aproveitando do argumento do limite de 20MB para envio de arquivos pelo TCCMensagens, o agente malicioso divulga aos membros do grupo um link para acesso a uma página externa para download dos cursos piratas. Porém, além do conteúdo esperado, a página também possui um código malicioso para inclusão do arquivo JavaScript dinâmico e envio de seus dados para o atacante. Por sua divulgação através do próprio TCCMensagens, o atacante verifica a forte eficácia obtida, sendo possível o acesso ao arquivo com dados de diversas pessoas.

Neste caso, o problema poderia ter sido evitado. Primeiramente, a separação do código JavaScript e dos dados do usuário deveria ser uma preocupação por parte da equipe de desenvolvimento do mensageiro, evitando a existência de arquivos de código dinâmicos. Além disso, para o problema da existência da API, pode-se propor duas soluções:

- Armazenamentos dos dados em um mesmo domínio da aplicação, evitando requisições cruzadas e pemitindo uma política não permissiva para o atributo *SameSite*;

- A utilização de outros mecanismos para a realização de requisições cruzadas, como o CORS "*Cross-Origin Resource Sharing*", por exemplo, declarando uma política restrita e com uma lista de permissões que permita apenas requisições direcionadas à API, aliado de uma política não permissiva para o atributo *SameSite* do *cookie* de sessão (MOZILLA, 2023b).

Por parte do usuário, as recomendações são similares ao cenário anterior: cuidados típicos contra engenharia social e utilização de um navegador que não permite a ocorrência de *Cross-Site Script Inclusion*.

5 CONCLUSÃO

Com este trabalho, foi possível observar e concluir que, ao longo dos anos, conceitos relacionados com a vulnerabilidade *Cross-Site Script Inclusion* se modificaram e evoluíram, de acordo com as tecnologias relacionadas com as requisições de origem cruzada.

A implementação do atributo *SameSite* aos *cookies* por parte dos navegadores representou forte impacto na vulnerabilidade como um todo, modificando desde os cenários de exploração, até as estratégias de mitigação para o problema, demonstrando que uma configuração adequada do atributo pode representar a principal estratégia de combate.

Além disso, a maneira como os navegadores implementam a Política de Mesma Origem e o comportamento de *cookies* em requisições cruzadas não segue um padrão definido, havendo mudanças significativas de comportamento a depender do navegador, de maneira a impactar diretamente nos cenários de exploração da vulnerabilidade. Em termos práticos, o usuário hoje possui um maior controle das ferramentas necessárias para evitar a ocorrência da *Cross-Site Script Inclusion*, podendo optar por navegadores que possuem uma implementação mais restrita quanto à anexação de *cookies* em requisições de origem cruzada.

A literatura publicamente disponível sobre o assunto, de maneira generalista, se encontra desatualizada e não adequada quanto a mudanças cruciais ocorridas ao longo dos últimos anos, não englobando alterações na implementação de mecanismos de segurança relacionados.

Além dos cenários clássicos encontrados na literatura, a exploração de *Cross-Site Script Inclusion* contra arquivos de mídia são possíveis e podem representar um impacto na segurança dos dados de usuários de aplicações vulneráveis.

Com este trabalho, afirma-se que hoje a principal abordagem a ser utilizada como estratégia de mitigação à *Cross-Site Script Inclusion* é uma política não permissiva quanto ao atributo *SameSite* dos *cookies*, se tratando de uma im-

plementação do lado servidor. Hoje o lado cliente pode ter um papel ativo na mitigação do problema, utilizando navegadores que não permitem a ocorrência da vulnerabilidade.

5.1 Trabalhos futuros

A partir das conclusões obtidas por este trabalho, é possível definir diversos outros temas de pesquisa que podem se converter em trabalhos futuros. Cita-se como exemplos:

- Com as alterações observadas nos anos recentes quanto a mecanismos de segurança, pode-se atribuir como um trabalho futuro o estudo de como essas alterações podem impactar na prática outras vulnerabilidades clássicas, bem como a atualização do material disponível sobre;
- Como demonstrado, a inclusão de arquivos de mídia é possível e representa possível fonte de ameaça. Porém, um estudo mais aprofundado quanto a exploração destes casos específicos e de seu impacto é necessário para melhor compreensão do problema;
- Hoje, existem diversos mecanismos que definem o comportamento de requisições de origem cruzada, como por exemplo o CORS. A busca por outros mecanismos e os problemas de segurança relacionados com uma má implementação pode ser também um trabalho futuro;
- A análise do código fonte dos navegadores em que isso é possível, bem como de seus navegadores base, pode revelar informações importantes quanto à maneira como é implementada a política de mesma origem e o comportamento dos *cookies* em requisições de origem cruzada.

REFERÊNCIAS

AMAZON. **User Guide Amazon Lightsail for Research**. [S.l.], 2024. Disponível em: <https://docs.aws.amazon.com/pdfs/lightsail-for-research/latest/ug/lightsail-for-research.pdf>. Acesso em: mai. 2024.

APACHE. **Documentação do Servidor HTTP Apache Versão 2.4**. [S.l.], 2024. Disponível em: <https://httpd.apache.org/docs/2.4/>. Acesso em: mai. 2024.

CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANçA NO BRASIL. **Cartilha de segurança para internet**. [S.l.], 2012. Disponível em: <https://cartilha.cert.br/livro/cartilha-seguranca-internet.pdf>. Acesso em: mai. 2024.

DEBIAN. **The Debian Administrator's Handbook**. [S.l.], 2022. Disponível em: <https://www.debian.org/doc/manuals/debian-handbook/index.en.html>. Acesso em: mai. 2024.

DUCK DNS. **About DuckDNS**. [S.l.], 2024. Disponível em: <https://www.duckdns.org/about.jsp>. Acesso em: mai. 2024.

ELECTRONIC FRONTIER FOUNDATION. **About Certbot**. [S.l.], 2024. Disponível em: <https://certbot.eff.org/pt-br/pages/about>. Acesso em: mai. 2024.

FRANKEN, G.; GOETHEM, T. V.; JOOSEN, W. **Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies**. 27th USENIX Security Symposium, 2018. Disponível em: <https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-franken.pdf>. Acesso em: mai. 2024.

GIL, A. C. **Como elaborar projetos de pesquisa**. 6. ed. [S.l.]: Editora Atlas, 2017.

GOOGLE. **Gmail Help**. [S.l.], 2024. Disponível em: <https://support.google.com/mail/?hl=en#topic=7065107>. Acesso em: mai. 2024.

GROSSMAN, J. **Advanced Web Attack Techniques using GMail**. Blog Jeremiah Grossman, 2006. Disponível em: <https://blog.jeremiahgrossman.com/2006/01/advanced-web-attack-techniques-using.html>. Acesso em: mai. 2024.

HAILPERIN, V.; RUEF, M. **Cross-Site Script Inclusion A Fameless but Widespread Web Vulnerability Class**. [S.l.]: SCIP, 2016.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 1035 DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION**. [S.l.], 1987. Disponível em: <https://www.ietf.org/rfc/rfc1035.txt>. Acesso em: mai. 2024.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 1738 Uniform Resource Locators (URL)**. [S.l.], 1994. Disponível em: <https://datatracker.ietf.org/doc/html/rfc1738>. Acesso em: mai. 2024.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 1866 Hypertext Markup Language - 2.0**. [S.l.], 1995. Disponível em: <https://datatracker.ietf.org/doc/html/rfc1866>. Acesso em: mai. 2024.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 2616 Hypertext Transfer Protocol – HTTP/1.1**. [S.l.], 1999. Disponível em: <https://datatracker.ietf.org/doc/html/rfc2616>. Acesso em: mai. 2024.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 2818 HTTP Over TLS**. [S.l.], 2000. Disponível em: <https://www.ietf.org/rfc/rfc2818.txt>. Acesso em: mai. 2024.

INTERNATIONAL NETWORK WORKING GROUP. **RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files**. [S.l.], 2005. Disponível em: <https://www.ietf.org/rfc/rfc4180.txt>. Acesso em: mai. 2024.

INTERNET ENGINEERING TASK FORCE. **RFC 8259 The JavaScript Object Notation (JSON) Data Interchange Format**. [S.l.], 2017. Disponível em: <https://datatracker.ietf.org/doc/html/rfc8259>. Acesso em: mai. 2024.

KERN, C.; DASWANI, N.; KESAVAN, A. **Foundations of Security: What Every Programmer Needs to Know**. [S.l.]: Apress, 2007.

LEKIES, S.; ENGELS, D.; MITKOV, M. **JSONPS: Secure an inherently insecure practice with this one weird trick!** 2021 IEEE European Symposium on Security and Privacy Workshops (EuroSPW), 2021. Disponível em: <https://ieeexplore.ieee.org/document/9583723>. Acesso em: mai. 2024.

LEKIES, S. et al. **The Unexpected Dangers of Dynamic JavaScript**. 24th USENIX Security Symposium, 2015. Disponível em: <https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-lekies.pdf>. Acesso em: mai. 2024.

MICROSOFT. **Windows 11 overview**. [S.l.], 2024. Disponível em: <https://learn.microsoft.com/en-us/windows/whats-new/windows-11-overview>. Acesso em: mai. 2024.

MOZILLA. **Firefox ESR 102.1.0**. [S.l.], 2022. Disponível em: <https://www.mozilla.org/en-US/firefox/102.1.0/releasenotes/>. Acesso em: mai. 2024.

MOZILLA. **JavaScript**. [S.l.], 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: mai. 2024.

MOZILLA. **Cookies HTTP**. [S.l.], 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Cookies>. Acesso em: mai. 2024.

MOZILLA. **Cross-Origin Resource Sharing (CORS)**. [S.l.], 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS>. Acesso em: mai. 2024.

MOZILLA. **Same-origin policy**. [S.l.], 2023. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy>. Acesso em: mai. 2024.

MOZILLA. **Firefox ESR 115.8.0**. [S.l.], 2024. Disponível em: <https://www.mozilla.org/en-US/firefox/115.8.0/releasenotes/>. Acesso em: mai. 2024.

MOZILLA. **Set-Cookie**. [S.l.], 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/Set-Cookie>. Acesso em: mai. 2024.

NIKIFORAKIS, N. et al. **You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions**. 2012 ACM conference on Computer and communications security, 2012. Disponível em: <https://dl.acm.org/doi/10.1145/2382196.2382274>. Acesso em: mai. 2024.

NWEKE, L. O. **Using the CIA and AAA Models to Explain Cybersecurity Activities**. PM World Journal Vol. VI, 2017. Disponível em: <https://pmworldlibrary.net/wp-content/uploads/2017/05/171126-Nweke-Using-CIA-and-AAA-Models-to-explain-Cybersecurity.pdf>. Acesso em: mai. 2024.

ORACLE. **Oracle VM VirtualBox User Manual**. [S.l.], 2024. Disponível em: <https://download.virtualbox.org/virtualbox/7.0.16/UserManual.pdf>. Acesso em: mai. 2024.

OWASP. **Testing for Cross Site Script Inclusion**. [S.l.], 2020. Disponível em: <https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/13-Testing_for_Cross_Site_Script_Inclusion>. Acesso em: mai. 2024.

OWASP. **Cross Site Request Forgery (CSRF)**. [S.l.], 2023. Disponível em: <https://owasp.org/www-community/attacks/csrf>. Acesso em: mai. 2024.

STUTTARD, D.; PINTO, M. **The Web Application Hacker's Handbook**. 2. ed. [S.l.]: Wiley Publishing, 2011.

TERADA, T. **Identifier based XSSI attacks**. MBSD Technical Whitepaper, 2015. Disponível em: <https://www.mbsd.jp/Whitepaper/xssi.pdf>. Acesso em: mai. 2024.

APÊNDICE A – Código da página de login da aplicação vulnerável

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

    <!-- Definição do estilo -->
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }

        #login-container {
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            width: 300px;
            text-align: center;
        }

        input {
            width: 100%;
```

```
            padding: 10px;
            margin: 10px 0;
            box-sizing: border-box;
        }

        button {
            background-color: #4caf50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
    </style>
</head>
<body>

<!-- Criação da interface de login -->
<div id="login-container">
    <h2>Bem-vindo ao TCC XSSi Goat!</h2>
    <form id="login-form">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>

        <button type="button" onclick="login()">Login</button>
    </form>
</div>
```

```
<!-- Checagem de credenciais e criação do cookie -->
<script>
    function login() {
        var username = document.getElementById('username').value;
        var password = document.getElementById('password').value;

        if (username === 'tcc' && password === 'tcc') {
            document.cookie = "authenticated=true; expires=Fri, 31 Dec 9999 23:59:59 GMT;
            path=/; SameSite=None; Secure";

            window.location.href = "main.html";
        } else {
            alert('Credeciais incorretas! Tente: tcc / tcc');
        }
    }
</script>

</body>
</html>
```

APÊNDICE B – Arquivos vulneráveis

JSONP:

```
<!-- Definição dos dados e da função para recuperação -->
processarDados({
  "nome": "TCC",
  "idade": 2024,
  "senha": "Tcc123"
});
```

JavaScript:

```
<!-- Criação e definição de variáveis globais -->
var nome;
var idade;
var senha;

nome = "TCC";
idade = 2024;
senha = "Tcc123";
```

Imagem:

APÊNDICE C – Arquivo .htaccess na aplicação vulnerável

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

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

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

APÊNDICE D – Código fonte das páginas maliciosas na aplicação maliciosa

Exploração do *JSONP*:

```
<!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>

  <!-- Recuperação e exibição dos dados -->
  <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>

  <!-- Inclusão do JSONP externo -->
  <script src="https://tccxssi.duckdns.org/tcc.jsonp"></script>

</body>
</html>
```

Exploração do *JavaScript*:

```
<!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>

<!-- Inclusão do script externo -->
<script src="https://tccxssi.duckdns.org/tcc.js"></script>

<!-- Exibição das variáveis globais via console -->
<script>
  console.log("Nome:", nome);
  console.log("Idade:", idade);
  console.log("Senha:", senha);
</script>

</body>
</html>
```

Exploração da imagem:

```
<!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>
```

```
<!-- Inclusão da imagem ao documento -->
<img src="https://tccxssi.duckdns.org/tcc.jpg">
</body>
```