Exploração De Vulnerabilidades *Cross-Site Scripting*: Uma Análise Das Principais Técnicas De Ataques XSS

Lucas Matheus Gonçalves Faculdades Integradas de Itararé - FA-FIT
Sengés, Brasil
lucas.66matheus@gmail.com

Leticia Maria de Oliveira Camenar
Faculdades Integradas de Itararé - FA-FIT
Itararé, Brasil
leticia.camenar@gmail.com

***Abstract* — In this article, we seek to analyze attack techniques with XSS, demonstrating these attacks in the bWAPP test environment, through the main payloads found in the OWASP Project. Criteria are also presented to mitigate and correct the flaws found. The Cross-Site Scripting (XSS) attack method is one of the vulnerabilities frequently found in web applications, as it allows the injection of malicious scripts that, when executed, can cause serious problems, such as session hijacking and redirection to websites malicious.**

***Resumo ou Resumen* — Neste artigo, busca-se analisar técnicas de ataque com XSS, demonstrando esses ataques no ambiente de testes bWAPP, por meio dos principais *payloads* encontrados no Projeto OWASP. Apresentam-se também critérios para mitigar e corrigir as falhas encontradas. O método de ataque *Cross-Site Scripting* (XSS) é uma das vulnerabilidades frequentemente encontradas em aplicações *web*, uma vez que, permitem a injeção de *scripts* maliciosos que quando executados, podem acarretar graves problemas, como o sequestro de sessão e redirecionamento para *sites* maliciosos.**

***Palavras-chave* — Segurança da Informação; Injeção de código; JavaScript; Vulnerabilidades.**

I. INTRODUÇÃO

A falta de conhecimento de muitos desenvolvedores incentivou a chegada de usuários mal-intencionados que dedicam seu tempo *online* a explorar falhas em aplicações com o objetivo de se beneficiar e cometer cibercrimes, como por exemplo, o roubo de informações, propagação de *spams* e vírus e controle de máquinas zumbis.

Existem múltiplas formas de se explorar falhas em aplicações *web*. Uma das mais utilizadas é por meio de um método de ataque que realiza a injeção de *scripts* maliciosos, denominado *Cross-Site Scripting* (XSS).

O XSS age numa página *web*, podendo alterar o comportamento de uma aplicação sem que o usuário ou desenvolvedor perceba (OWASP, 2021a). Esta prática dá ao atacante a possibilidade de ler todo o conteúdo da página e de mandar informações para um servidor controlado pelo atacante, violando assim, o conceito de privacidade de uma aplicação.

O OWASP, apresenta o método de ataque XSS como uma das vulnerabilidades frequentemente encontradas em aplicações *web* e argumenta que se esse método de injeção de *scripts* for explorado profundamente, pode acarretar graves problemas para a aplicação (OWASP, 2021b).

A exploração das vulnerabilidades XSS, pode ocorrer em três principais categorias: refletido (não persistente), armazenado (persistente) e o *Document Object Model* (DOM) *Based* (J. SILVA, 2009).

A. *Problematização e Justificativa*

Estima-se que cerca de 40% de todos os ciberataques em 2019 foram performados utilizando *Cross-Site Scripting*, que é o vetor de ataque favorito dos *hackers* em todo o mundo (ILIC, 2019). Os ataques XSS tem como objetivo fazer com que o atacante se passe pela vítima ou que ocorra um sequestro de sessão, no entanto, ataques mais sofisticados podem fazer com que a aplicação seja infectada por uma máquina zumbi (GROSSMAN et al, 2007).

Sabendo-se que os métodos de ataque *Cross-Site Scripting* (XSS) representam uma parcela considerável dos ataques frequentemente executados em páginas *web*, é interessante que os desenvolvedores possam ter acesso a documentações que apresentem os tipos de ataques que possam ser injetados em suas aplicações e mostrem de que forma podem ser implementados métodos de defesa aos ataques XSS (ACUNETIX, 2020).

O presente artigo visa entender de que forma as aplicações *web* pode ser exploradas por meio de ataques XSS e que diretrizes devem ser tomadas para mitigar as falhas ocorridas pela injeção de códigos maliciosos em aplicações *web*.

Por ser um estudo da área da computação, este artigo possui como referência metodológica o livro de Waslawick (2009), que aborda a metodologia de pesquisa em trabalhos na área da ciência da computação. Desta forma, a natureza desta pesquisa é aplicada, uma vez que, buscamos entender as particularidades dos ataques *Cross-Site Scripting* em um contexto de aplicações *web*. Também podemos classificá-la como um trabalho exploratório, que tem uma abordagem qualitativa, já que, serão analisados os principais tipos de ataque XSS, por meio de uma ferramenta de teste e em conclusões obtidas pelos autores tendo como base a literatura, a observação e a singularidade do ataque XSS. Para obter estes resultados, optou-se por realizar procedimentos comuns às pesquisas experimentais, já que faremos os testes propostos

em um ambiente controlado e que condiz com os resultados esperados.

B. *Objetivos*

1) *Objetivo Geral*

O artigo proposto tem como objetivo geral analisar as principais técnicas de ataque *Cross-Site Scripting* (XSS), demonstrando na prática como explorar e mitigar essas falhas.

2) *Objetivos Específicos*

- Descrever métodos de ataque *Cross-Site Scripting* (XSS), apresentando suas especificações;
- Demonstrar na prática como realizar ataques XSS e quais as consequências dessas vulnerabilidades;
- Apresentar modos de se proteger e mitigar a injeção de código malicioso em aplicações *web*;

3) *Organização do trabalho*

Este artigo está organizado da seguinte forma: a primeira seção apresenta a introdução, contendo a problematização, justificativa e objetivos que permeiam este estudo. A segunda seção discorre sobre o referencial teórico dessa pesquisa, apresentando conceitos de segurança da informação, tipos de ataque *Cross-Site Scripting* e vulnerabilidades. A terceira seção descreve os materiais e métodos utilizados formular os testes de vulnerabilidades. A quarta seção apresenta os resultados obtidos com os testes e as discussões pertinentes. Por fim, apresentam-se as considerações finais.

II. Referencial Teórico

A. *Segurança da Informação*

A Segurança da Informação pode ser entendida como um processo que visa a proteção de dados informacionais, garantindo assim a integridade, disponibilidade e confiabilidade dos dados (BEAL, 2005). Garantir esses três conceitos é um processo complexo para projetistas, analistas desenvolvedores e profissionais de tecnologia da informação, uma vez que, deve-se levar em conta “O que você está tentando proteger? Por que você está tentando proteger? Como você vai protegê-lo?” (RHODES-OUSLEY, 2013).

Estes questionamentos são constantes em ambientes de desenvolvimento, já que a Segurança da Informação se apresenta como um importante pilar dentro dos sistemas computacionais. No entanto, existem empresas e desenvolvedores que ainda optam por fazer com que a segurança de seus sistemas seja um fator opcional, preocupando-se mais com a minimização do tempo e com a diminuição do custo de desenvolvimento dos projetos.

Essa decisão de considerar a segurança como opcional, pode ser um fator crítico para gerar falhas que podem ser usadas para obter e alterar informações. Ao alterar essas informações, um indivíduo mal-intencionado pode roubar dados sigilosos, interceptar comunicações entre cliente e servidor, parar ou no pior dos cenários, destruir o serviço *web*, deixando-o inutilizável ou sem acesso (LUZ, 2011).

Uma vulnerabilidade muito explorada por potenciais invasores é a *Cross-Site Script* (XSS), que será descrita em detalhes a seguir.

B. *Cross-Site Scripting Refletido (Não Persistente)*

A vulnerabilidade do tipo XSS Refletido é caracterizado por receber os dados não seguros de um usuário e que reflete diretamente para o *browser* (DADARIO, 2012). Para isso acontecer, o usuário precisa acessar a URL construída pelo invasor, onde o servidor vai repetir no código fonte da aplicação o *script* malicioso (PORTSWIGGER, 2021).

O método mais comum de realizar esse tipo de ataque, é através do uso de engenharia social, onde o atacante pode usar *phishing* em *e-mails* ou redes sociais para induzir as vítimas a carregar a URL com o *payload* (OWASP, 2021a). Baseado na documentação do Projeto OWASP, podemos ver na Figura 1, o ambiente de testes bWapp (MESELLEM, 2021) com a vulnerabilidade XSS Refletido.

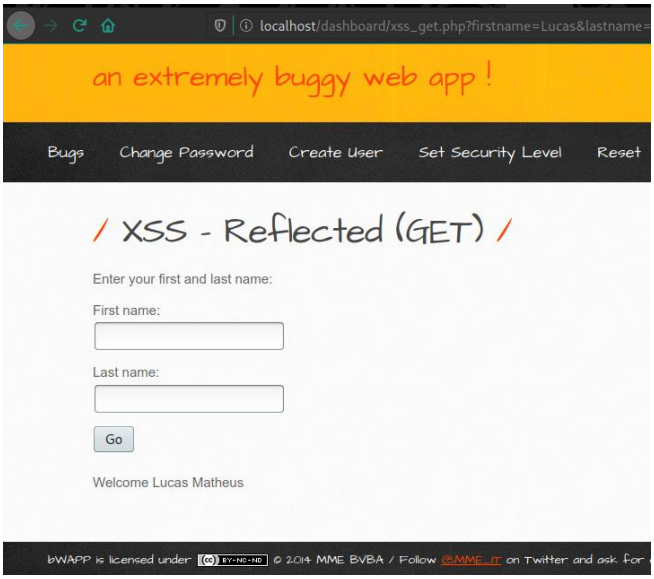

Fig. 1. Tela de entrada de dados com a carga maliciosa executando.

Os dados inseridos nos campos de texto *First name* e *Last name* são refletidos na tela, pelo texto "*Welcome* Lucas Matheus". Mesmo que o texto não fosse exibido, podemos analisar que os dados são passados na URL na aplicação:

http://localhost/dashboard/xss_get.php?firstname=Lucas&lastname=Matheus&form=submit

Trocando o dado "Matheus" pelo *payload*:

<iframe src="https://jcw87.github.io/c2-smb1/" width="100%" height="500"></iframe>

A figura 2 mostra como podemos refletir o jogo Super Mário Bros, que está armazenado no GitHub:

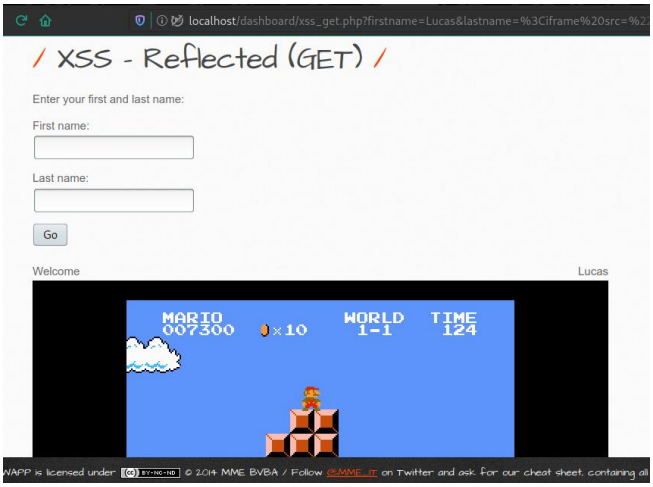

Fig. 2. Tela de entrada de dados com a carga maliciosa executando.

C. *Cross-Site Scripting Armazenado (Persistente)*

O XSS persistente ou armazenado ocorre quando o código malicioso é armazenado no servidor da aplicação e em seguida é recuperado quando a vítima solícita essas informações armazenadas. Esse formato de exploração de XSS, é o mais perigoso, pois o ataque se repetirá automaticamente sempre que a informação for recuperada e exibida para o usuário, sendo mais recorrente em redes sociais, *blogs* ou campos de comentários que possam ser lidos por outros usuários (OWASP, 2021a).

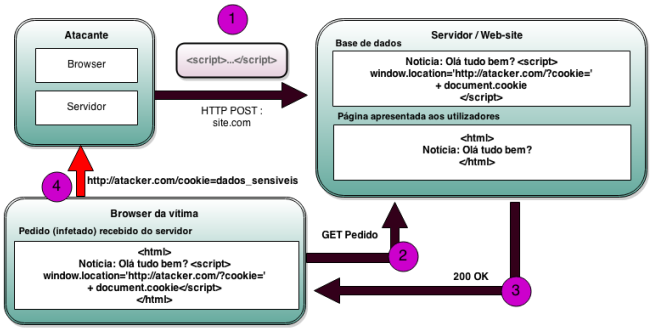

Fig. 3. Fluxo de um ataque XSS.

Através da vulnerabilidade XSS persistente podemos fazer o sequestro dos *cookies* de um usuário com o auxílio do utilitário Netcat (GIACOBBI, 2006) através do comando, MARIANO (2019):

$ netcat -nvlp 9000

-n define a porta

-v realiza a saída de dados detalhado

-l especifica o tamanho do buffer de recebimento do TCP.

-p especifica a porta de origem

Na página vulnerável podemos usar uma carga *script* maliciosa, que redireciona os *cookies* do usuário para o servidor do atacante:

<script>new Image().src="http://servidordoatacante:9000/="+document.cookie</script>

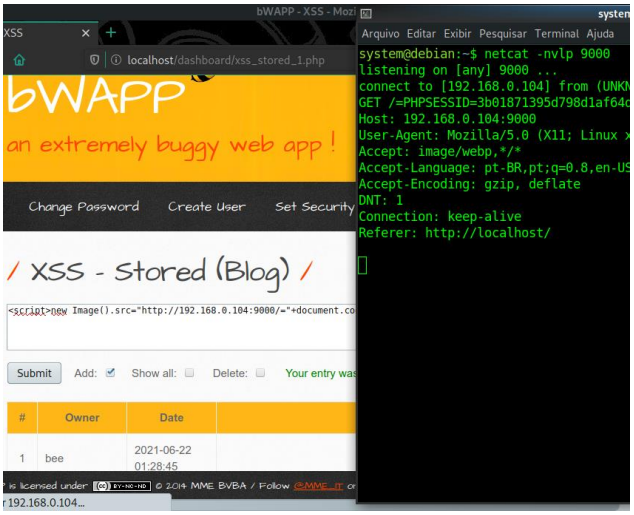

Fig. 4. *Payload* de XSS para roubo de *cookie*.

Em um cenário real, Endler (2002) cita que um dos maiores obstáculos para um invasor transformar um *exploit* (em português, exploração) XSS de roubo de *cookies* em uma exploração bem-sucedida depende do momento exato que que a vítima estiver fazendo acesso a página infectada.

D. *Cross-Site Scripting Baseado em Dom (Modelo De Objeto De Documentos)*

O *Document Object Model* (DOM) é uma especificação adotada pela W3C (*World Wide Web Consortium*), com objetivo de padronizar a estrutura de uma página independente da forma que está sendo interpretado no navegador *web*. (DADARIO, 2021). Essa especificação, define o termo "documento" para a linguagem de marcação, sendo representado em uma árvore de nós como na Figura 5.

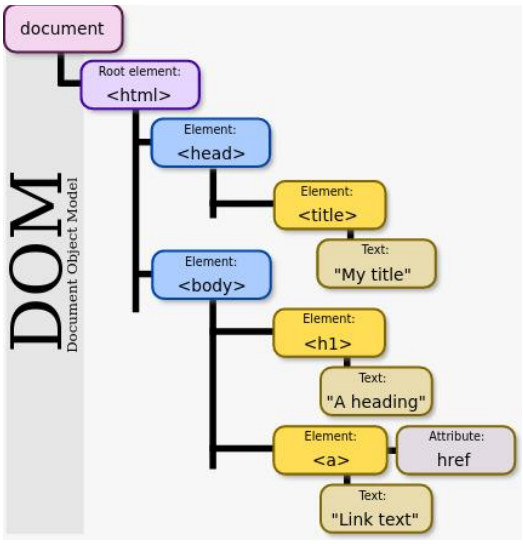

Fig. 5. Estrutura do Document Object Model - DOM.

O *Javascript* tem um papel comum na modificação da árvore DOM em uma página *web*, pois qualquer coisa criada no conteúdo da página pode ser acessada através do objeto *document* (MEDEIROS, 2013). Se existir falhas no interpretador de *script* do navegador ou entradas de dados sem validação, é possível realizar o ataque XSS Baseado em DOM, efetuando as modificações da página *web* sem a interação com o servidor onde está hospedado (REDE SEGURA, 2012).

Explorar manualmente falhas no DOM de uma página *web* é um trabalho árduo, pois é necessário que o desenvolvedor faça análises dos parâmetros do código fonte da página (OWASP, 2021c).

E. Mitigação Das Vulnerabilidades Cross-Site Scripting

Segundo Grossman et al. (2007) as vulnerabilidades de XSS são um problema complexo, pois podem contextualizar com *bugs* do navegador e sua incapacidade de decidir se determinado código é pertencente a página de origem ou se foi injetado. Outro problema, é o desafio dos desenvolvedores na construção de *sites* seguros, permitindo que os invasores explorem as falhas destes *sites.*

Uma medida que se tornou eficaz contra os ataques XSS, é o uso do mecanismo de segurança CSP (*Content Security Policy*) nos navegadores de *internet*. Patil (2017) explica que o CSP fornece diretrizes de restrição de conteúdo diretamente no cabeçalho HTTP, dando permissão de executar somente o que estiver em uma ementa de conteúdos confiáveis criado pelos desenvolvedores da aplicação *web* ou através de uma lista predefinida.

"Um navegador compatível com o CSP só irá executar então *scripts* que vierem de arquivos que estejam presentes nos domínios que foram previamente especificados como confiáveis" (MOZILLA DEVELOPER, 2021).

Também é possível fazer a filtragem das *tags* da aplicação através do método de sanitização do HTML, impedindo que códigos maliciosos sejam carregados para o código original. O HTML *Purifier* (http://htmlpurifier.org/) é uma biblioteca *open-source* que demonstra uma grande capacidade em preservar o conteúdo legítimo da página fazendo uma análise das entradas de dados com uma lista especificada pelo administrador do *site*.

III. MATERIAIS E MÉTODOS

Os ataques do tipo *Cross-Site Scripting* consistem na modificação da estrutura de aplicações *web*, o que pode ser considerado como crime de dano ao patrimônio de acordo com o Art. 163 do Código Penal Brasileiro (BRASIL, 1940). Por essa razão, optamos pelo uso de uma aplicação *open-source* exclusiva para que profissionais na área da segurança da informação possam testar diversas vulnerabilidades, para que fizéssemos os testes apresentados neste artigo.

O bWAPP (http://www.itsecgames.com) é uma aplicação desenvolvida em PHP e que utiliza o SGBD MySQL, podendo ser hospedado em ambientes GNU/Linux, Windows e macOS. O diferencial dessa ferramenta perante suas concorrentes, é sua facilidade de instalação e configuração, além disso, ela foi projetada para cobrir todas as vulnerabilidades do projeto OWASP Top 10 (OWASP, 2021d).

A estrutura usada na hospedagem e acesso do ambiente de testes bWAPP foi detalhado na Tabela 1:

TABELA I
ESTRUTURA USADO PARA HOSPEDAR E ACESSAR O BWAPP.

Sistema operacional	GNU/Linux Debian 10 (buster) 64-bit
Processador	Intel Core I5 2.3GHz
Memória RAM	8 GB DDR3
Pacote de servidor	XAMPP
Navegador web	Mozilla Firefox

Processo de instalação do XAMPP no Debian:

1° Efetuar o *download* do XAMPP: www.apachefriends.org;

2° Abrir o terminal Linux;

3° Dar permissão de execução do arquivo no Linux:

chmod 777 xampp-linux-x64-7.3.28-1-installer.run;

4° Executar a instalação:

sudo ./xampp-linux-x64-7.3.28-1-installer.run;

Processo de instalação do bWAPP no XAMPP:

1° Efetuar o *download* do bWAPP: www.itsecgames.com;

2° Extrair o arquivo bWAPP_latest.zip no diretório /opt/lampp/htdocs;

3° Iniciar o servidor XAMPP com o comando:

sudo /opt/lampp/xampp start;

4° Acessamos bWAPP no navegador pela URL: http://localhost/dashboard/login.php;

5° Por padrão é definido o *Login* como: *bee* e o *Password* como: *bug*, permitindo o acesso como podemos ver na Figura 6.

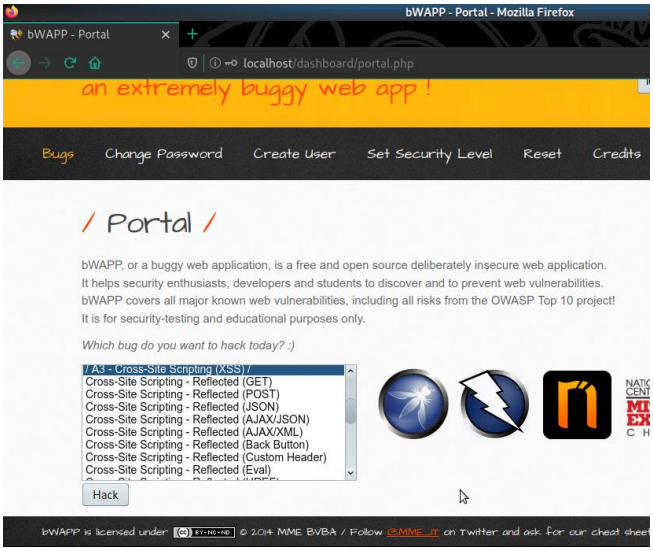

Fig. 6. Aplicação de testes de vulnerabilidades bWAPP.

Existem diversas formas de fazer injeções de códigos maliciosos, onde cada método se adapte ao contexto da vulnerabilidade. Na seção II por exemplo, foi apresentado na prática duas formas de explorar o XSS, para modificar a estrutura exibindo o jogo Super Mário Bros e para realizar o sequestro de *cookies* do usuário que visitasse a página.

Na Figura 7 a codificação da página apresenta uma brecha por falta de fechamento das chaves, permitindo um XSS Refletido. Inserindo o payload: "}]}';alert("XSS");</script> a página retorna um *alert* com uma mensagem como na Figura 8.

```
    <script>

        var JSONResponseString = '{"movies":[{"response":"HINT: our master
                  really loves Marvel movies :)"}]}';
        // var JSONResponse = eval ("(" + JSONResponseString + ")");
        var JSONResponse = JSON.parse(JSONResponseString);

        document.getElementById("result").innerHTML=JSONResponse.movies[0
                  ].response;
    </script>
</div>
```

Fig. 7. Código JSON com vulnerabilidade XSS Refletido.

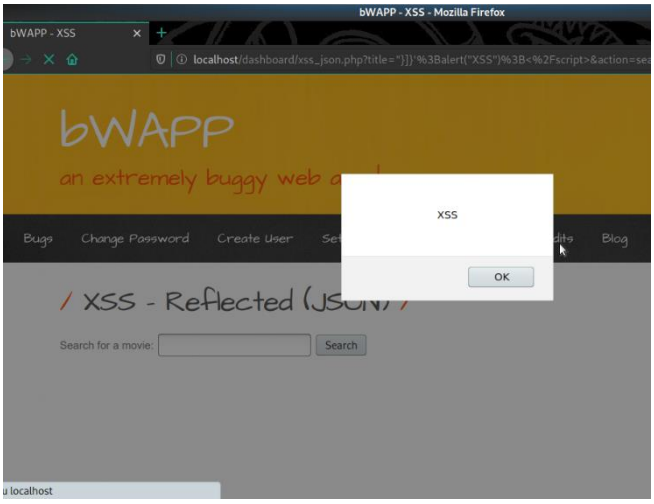

Fig. 8. Mensagem sendo refletido na página.

Também é possível usar os recursos do CSS ou HTML da página para injetar uma carga maliciosa, mesclando o *JavaScript* em sua estrutura:

```
<style>
    div  {
        background-image:
url("<\/style><svg/onload=alert(docu
ment.domain)>");
        background-color: #cccccc;
    }
    </style>
```

O CSP como apresentado na seção 2, é um dos melhores métodos para prevenir o *Cross-Site Scripting*, pois permite que programadores de aplicações para a internet declarem a origem das informações enviadas e requisitadas pelos usuários. Por ser um dos melhores, é o método frequentemente usado para mitigar ataques XSS.

Para usar o CSP é necessário que o navegador ofereça suporte para o recurso e que o desenvolvedor configure na página *web* o cabeçalho *Content-Security-Policy* com seus valores para controlar os recursos fornecidos no conteúdo da aplicação web, dificultando a injeção de códigos maliciosos (SANTOS, 2020). Na Figura 9, podemos ver o CSP implantado nas páginas PHP do bWAPP como forma de impedir que códigos XSS fossem refletidos ou armazenados pela aplicação.

```php
<?php

// Regras em formato de array para ficar fácil de manter
$rules = [
    'script-src' => [
        "'self'",
        'https://google.com',
    ],
    'style-src' => [
        "'self'",
    ],
];
$header = 'Content-Security-Policy: ';
foreach ($rules as $directive => $values) {
    $header .= sprintf('%s %s;', $directive, implode(' ', $values));
}
$header = trim($header, ';');
header($header);

/*

bWAPP, or a buggy web application, is a free and open source deliberatel
It helps security enthusiasts, developers and students to discover and t
bWAPP covers all major known web vulnerabilities, including all risks fr
It is for security-testing and educational purposes only.
```

Line 20, Column 3

Fig. 9. CSP — *Content Security Policy* usado em código PHP.

IV. CONCLUSÃO

A difusão da internet mundial permitiu a facilidade do dia a dia com as variedades de aplicações, porém assegurar a confiabilidade dos dados dos usuários vem se tornando uma tarefa complexa devido aos cibercrimes. É impossível existir um sistema que seja totalmente seguro, porém com implementações de *firewalls*, análise e manutenção dos códigos que pode apresentar riscos de instabilidade, são táticas importantes para assegurar aos usuários segurança de navegação.

Entre as diversas vulnerabilidades existentes, o *Cross-Site Scripting* é um método de explorar o código fonte de uma aplicação *web* para que sejam injetados scripts maliciosos (normalmente em *JavaScript*) e realizar alguma modificação de sua estrutura. Apesar de parecer uma simples vulnerabilidade, se um atacante explorar indevidamente o código fonte de uma página, podemos ter uma fonte primária para outras formas de ataque, como o *Session Hijacking* (sequestro de sessão) ou o *Phishing*.

Os testes foram limitados em uma máquina local, porém é possível entender um cenário mais real de um servidor na internet com o uso de máquinas virtuais, permitindo criar servidores para testar a complexidade das aplicações em eventuais vulnerabilidades. Entre algumas ferramentas para automatizar o processo de varredura do código em busca de falhas, poucos demonstraram sucesso por essa restrição durante a pesquisa.

Esse artigo permite que sejam realizadas eventuais pesquisas explorando o XSS com outros ataques como o SQL *injection*, CSRF (*Cross-site Request Forgery*), *man-in-the-middle* e *Session hijacking*. Além de possibilitar uma visão para que desenvolvedores *web* cometam menos falhas durante a codificação de aplicativos e sites e busquem formas de mitigar possíveis ataques.

REFERÊNCIAS

[1] ACUNETIX. Acunetix Web Application Vulnerability Report 2020. Disponível em: <https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2020/>. Acesso em: 15.mai.2021.

[2] BEAL, Adriana. Conceitos e Princípios Básicos da Informação. Segurança da Informação. São Paulo: Atlas, 2005.

[3] BRASIL. Decreto-Lei n. 2.848, de 7 de dezembro de 1940. Decreta a lei do Código Penal. Rio de Janeiro, 7 de dezembro de 1940. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto-lei/del2848.htm>. Acesso em 15.jun.2021

[4] DADARIO, Anderson. Anderson Dadario. XSS – Guia explicativo, 2012. Disponível em <https://dadario.com.br/xss-guia-explicativo/>. Acesso em 15.jun.2021.

[5] ENDLER, David. The Evolution of Cross-Site Scripting Attacks. Technical Report, iDEFENSE Inc., 2002.

[6] ERIKSSON, Birger. Wikimedia Commons. DOM-model, 2012. Disponível em: <https://commons.wikimedia.org/wiki/File:DOM-model.svg>. Acesso em: 26.jun.2021.

[7] GIACOBBI, Giovanni. Netcat. The GNU Netcat project, 2006. Disponível em: < http://netcat.sourceforge.net/>. Acesso em: 16. jun.2021.

[8] GROSSMAN J, HANSEN R, PETKOV P, RAGER A, FOGIE S (org.). Cross Site Scripting Attacks: XSS Exploits and Defense. United States of America: Elsevier, 2007. 482 p.

[9] ILIC, J. Cross-Site Scripting (XSS) Makes Nearly 40% of All Cyber Attacks in 2019. Disponível em <https://www.precisesecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/>. Acesso em: 29.jul.2021.

[10] J. SILVA, Armando Gonçalves. Cross-Site Scripting: Uma Análise Prática. Universidade Federal de Pernambuco, 2009.

[11] KIRSTEN, S. et al. OWASP. Cross Site Scripting (XSS). Disponível em: <https://owasp.org/www-community/attacks/xss/>. Acesso em: 15.jun.2021.

[12] LUZ, H. J. F. Análise de Vulnerabilidades em Java Web Applications. Trabalho de Conclusão de Curso. 2011. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha. Marília, São Paulo.

[13] MARIANO, Marcio. Linux Force Security. Comandos Linux – Comando nc, 2019. Disponível em: <https://www.linuxforce.com.br/comandos-linux/comandos-linux-comando-nc/>. Acesso em: 20.jun.2021.

[14] MEDEIROS, Higor. DevMedia | Plataforma para Programadores, Trabalhando com DOM em JavaScript, 2013. Disponível em: <https://www.devmedia.com.br/trabalhando-com-dom-em-javascript/29039>. Acesso em: 17.jun.2021.

[15] MESELLEM, M. bWAPP – An extremely buggy web app. Disponível em <http://www.itsecgames.com/> Acesso em: 15.jun.2021.

[16] MOZILLA DEVELOPER. Utilizando Políticas de Segurança de Conteúdo, 2021 Disponível em:< https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CSP/> Acesso em: 15.jun.2021.

[17] OWASP. Cross Site Scripting (XSS). Disponível em: <https://owasp.org/www-community/attacks/xss/> Acesso em: 01.jun.2021a.

[18] OWASP. Who is the OWASP® Foundation?. Disponível em <https://owasp.org/> Acesso em: 18.mai.2021b.

[19] OWASP. DOM Based XSS. Disponível em: <https://owasp.org/www-community/attacks/DOM_Based_XSS/>. Acesso em: 18.mai.2021c.

[20] OWASP. OWASP Top Ten. Disponível em: <https://owasp.org/www-project-top-ten/>. Acesso em: 18.mai.2021d.

[21] PATIL, Kailas. An Insecure Wild Web: A Large-Scale Study Of Effectiveness Of Web Security Mechanisms. Vishwakarma Institute of Information Technology, Pune, 2017.

[22] PORTSWIGGER. Cross-site scripting, 2021. Disponível em <https://portswigger.net/web-security/cross-site-scripting>. Acesso em 09.jun.2021.

[23] RHODES-OUSLEY, M. Information Security The Complete Reference. 2th. The McGrawHill Companies, 2013. 833.

[24] REDE SEGURA. O Ataque Cross-Site Scripting (XSS). 2012. Disponível em: < http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5680/material-seg-redes/Serie%20Ataques-RedeSegura-XSS.pdf>. Acesso em: 26.jun.2021.

[25] SANTOS, Josiel. O que é HTTP Content-Security-Policy e como utilizar no PHP, 2020. Disponível em: <https://jssantos.net/2020/04/o-que-e-http-content-security-policy-e-como-utilizar-no-php/?doing_wp_cron=1624581178.3552479743957519531250>. Acesso em: 28.jun.2021.

[26] SILVEIRA, Lucidia A. et al. Engenharia Social: Uma análise sobre o ataque de Phishing. Universidade Federal do Pamp, Rio Grande do Sul, 2007.

[27] SINGH, M, SINGH, P, KUMAR, P. An Analytical Study on Cross-Site Scripting, 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020, pp. 1-6, doi: 10.1109/ICCSEA49143.2020.9132894.

[28] TAVARES, Pedro. Segurança Informática. A bíblia do Cross-site Scripting (XSS), 2014. Disponível em: <https://seguranca-informatica.pt/a-biblia-do-cross-site-scripting-xss/>. Acesso em: 02.jun.2021.

[29] VOGT, Philipp. NENTWICH, Florian. JOVANOCIC, E. Kirda. KRUEGEL Chistopher. VIGNA, Giovanni. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. University of California, Santa Barbara, 2007.

[30] WASLAWICK, R. Metodologia de pesquisa para Ciência da Computação. 6. ed. Rio de Janeiro: Elsevier, 2009.