



FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Segurança da Informação

Luã Rapelli Boni

Injeção de código malicioso em aplicações Web
Uma análise do Cross Site Scripting

Americana, SP

2018



FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Segurança da Informação

Luã Rapelli Boni

Injeção de código malicioso em aplicações Web
Uma análise do Cross Site Scripting

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do Prof. Esp. Marcus Vinicius Lahr Giraldi.

Área de concentração: Análise de vulnerabilidades.

Americana, SP

2018

**FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte**

B698i BONI, Luã Rapelli

Injeção de código malicioso em aplicações Web: uma análise do Cross Site Scripting. / Luã Rapelli Boni. – Americana, 2018.

62f.

Monografia (Curso de Tecnologia em Segurança da Informação) - -
Faculdade de Tecnologia de Americana – Centro Estadual de Educação
Tecnológica Paula Souza

Orientador: Prof. Esp. Marcus Vinicius Lahr Giraldi

1 Segurança em sistemas de informação 2. Web – rede de computadores I. GIRALDI, Marcus Vinicius Lahr II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.518.5




Faculdade de Tecnologia de Americana


Injeção de Código Malicioso em Aplicações Web Uma análise do Cross Site Scripting

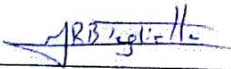
Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana.
Área de concentração: Análise de vulnerabilidades.

Americana, 25 de Junho de 2018.

Banca Examinadora:


Marcus Vinicius Lahr Giraldi (Presidente)
Especialista
Fatec Americana


Edson Roberto Gaseta (Membro)
Especialista
Fatec Americana


Marcio Roberto Baldo Taglietta (Membro)
Especialista
Fatec Americana

AGRADECIMENTOS

Em primeiro agradeço a minha querida esposa Barbara pelo apoio e compreensão durante as horas despendidas na execução deste trabalho. Também agradeço ao meu orientador Marcus Vinicius Lahr Giraldi pelo auxílio e orientações dadas a mim para que eu pudesse executar este trabalho da melhor maneira possível.

DEDICATÓRIA

Dedico este trabalho ao meu querido pai João Romildo Boni (em memória), falecido em 22/05/2018. Agradeço com todo meu amor e carinho, por toda dedicação com que venceu uma série de obstáculos para proporcionar o melhor para minha vida. Foi meu melhor amigo e sempre esteve ao meu lado torcendo pelo meu sucesso e vibrando por todas as minhas conquistas. Espero poder ter sido digno de todo esforço dedicado por você em todos os aspectos, em especial à minha formação.

RESUMO

Devido ao grande número de aplicações web e o crescimento exponencial de usuários na Internet utilizando serviços de nuvem para o armazenamento de dados sigilosos, é de extrema importância que se volte a atenção para segurança dos dados que entram nas aplicações web.

O presente trabalho tem por objetivo desenvolver uma análise das vulnerabilidades de aplicações web que permitem a prática de ataques de XSS (*Cross Site Scripting*). Estes ataques são realizados com a intenção de explorar fragilidades de entrada e saída de dados nas aplicações web que não são tratadas corretamente pelo desenvolvedor do sistema. A injeção destes códigos maliciosos é normalmente realizada utilizando a linguagem de programação *Javascript*.

Tais vulnerabilidades possibilitam que uma pessoa mal-intencionada venha furtrar informações sigilosas, redirecione o usuário para um site malicioso ou até mesmo execute ações dentro do sistema como um usuário autêntico.

Serão abordados no trabalho os conceitos sobre a vulnerabilidade, as principais técnicas de injeção de código e formas de prevenção. Fornecendo assim ao leitor uma visão abrangente do assunto, estando ele apto a aplicar as técnicas preventivas do XSS em qualquer aplicação web.

Este trabalho proporciona aos alunos e programadores um conhecimento amplo sobre os riscos que a ameaça do XSS pode causar nas aplicações web e a importância da aplicação de prevenções contra ela.

Palavras Chave: Cross Site Scripting (XSS), injeção de código, vulnerabilidade.

ABSTRACT

Due to the large number of web applications and the exponential growth of Internet users using cloud services for the storage of sensitive data, it is of the utmost importance to focus attention on the security of data entering web applications.

This work aims to develop a vulnerability analysis of web applications that allows the practice of attacks of XSS (Cross Site Scripting).

These attacks are intended to exploit weaknesses in data entry and exit in web applications that are not properly handled by the system developer. Injection of these malicious codes is usually done using the Javascript programming language.

Such vulnerabilities enables a malicious person to steal sensitive information, redirect the user to a malicious site, or even perform actions within the system as an authentic user.

The concepts of vulnerability, key code injection techniques and prevention approaches will be addressed in the paper. Providing the reader with a comprehensive view of the subject, being able to apply the XSS preventive techniques in any web application.

This work provides students and programmers with a comprehensive understanding of the risks that the XSS threat can cause in web applications and the importance of applying of prevention against it.

Keywords: *Cross Site Scripting (XSS), code injection, vulnerability.*

SUMÁRIO

1	INTRODUÇÃO	13
2	CONCEITOS INTRODUTÓRIOS.....	17
2.1	SEGURANÇA DA INFORMAÇÃO	17
2.2	APLICAÇÕES WEB	17
2.3	AS LINGUAGENS DA WEB.....	18
3	O CROSS SITE SCRIPTING	21
3.1	CLASSIFICAÇÃO QUANTO AOS TIPOS DE XSS.....	23
3.1.1	XSS ARMAZENADO	24
3.1.2	XSS REFLETIDO	26
3.1.3	XSS BASEADO EM DOM	29
4	CODIFICAÇÕES.....	34
4.1	CODIFICAÇÃO HTML.....	34
4.1.1	CODIFICAÇÃO ENTIDADE.....	35
4.1.2	CODIFICAÇÃO DECIMAL.....	36
4.1.3	CODIFICAÇÃO HEXADECIMAL	37
4.2	CODIFICAÇÃO PERCENTUAL	38
5	ANÁLISE DE VULNERABILIDADE DE UMA APLICAÇÃO WEB AO XSS	40
5.1	PROCEDIMENTOS DE TESTE	40
5.2	EXECUÇÃO DO ATAQUE XSS REFLETIDO.....	40
5.2.1	EXPLORANDO A VULNERABILIDADE	40
5.2.2	FORMAS DE PREVENÇÃO.....	43
5.3	EXECUÇÃO DO ATAQUE XSS ARMAZENADO.....	44
5.3.1	EXPLORANDO A VULNERABILIDADE	44
5.3.2	FORMAS DE PREVENÇÃO.....	49

5.4	EXECUÇÃO DO ATAQUE XSS BASEADO EM DOM.....	52
5.4.1	EXPLORANDO A VULNERABILIDADE	52
5.4.2	FORMAS DE PREVENÇÃO	55
6	PREVENÇÕES AO ATAQUE XSS	57
6.1	PRUDÊNCIA NA INSERÇÃO DE DADOS SENSÍVEIS.....	57
6.2	UTILIZAÇÃO DE CODIFICAÇÕES NO DESENVOLVIMENTO DE APLICAÇÕES	57
6.3	UTILIZAÇÃO DE BIBLIOTECAS PARA CODIFICAÇÃO SEGURA.....	59
7	CONSIDERAÇÕES FINAIS	60

LISTA DE ABREVIATURAS E SIGLAS

ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
PHP	PHP Hypertext Preprocessor
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
XML	eXtensible Markup Language
XSS	Cross Site Scripting

LISTA DE FIGURAS

Figura 1 - Total de incidentes reportados ao CERT.br por ano	13
Figura 2 - Estatística da ocorrência do XSS 2017	14
Figura 3 - Camadas de uma aplicação web	19
Figura 4 - Código inseguro que escreve tags no documento HTML	21
Figura 5 - Código inseguro que adiciona um conteúdo em um elemento HTML.....	21
Figura 6 - Ataque XSS utilizando o atributo onload da tag <body> do HTML	22
Figura 7 - Ataque XSS utilizando os atributos onmouseover e onerror	22
Figura 8 - Tipos de XSS de acordo com ambiente afetado	23
Figura 9 - Tipos de XSS de acordo com o nível de criticidade	24
Figura 10 - Formulário sem validação XSS armazenado	25
Figura 11 - Sistema infectado pelo XSS armazenado	26
Figura 12 - Código vulnerável ao XSS refletido.....	27
Figura 13 - Exemplo de busca com parâmetro pela URL.....	27
Figura 14 - Exemplo de busca com script malicioso	28
Figura 15 - Exemplo XSS refletido em uma loja virtual	28
Figura 16 - Exemplo de página infectada com XSS refletido em uma loja virtual	29
Figura 17 - Código HTML para demonstração do DOM.....	30
Figura 18 - Manipulação de elementos HTML através do DOM.....	31
Figura 19 - Exemplo de código XSS baseado em DOM	32
Figura 20 - Ataque XSS sem codificação	35
Figura 21 - Ataque XSS com codificação entidade	36
Figura 22 - Ataque XSS com codificação decimal.....	37
Figura 23 - Ataque XSS com codificação hexadecimal.....	37
Figura 24 - Código para demonstração da codificação de URL.....	39
Figura 25 - Exemplo de codificação percentual.....	39
Figura 26 - Injeção de código na URL através de codificação percentual.....	39
Figura 27 - Página de produtos	41
Figura 28 - Código de processamento da busca.....	41
Figura 29 - Script malicioso campo busca.....	42
Figura 30 - Resultado da busca com script malicioso	42
Figura 31 - Código de processamento da busca com filtro	43
Figura 32 - Resultado da busca com filtro aplicado.....	44

Figura 33 - Página de cadastro de usuários.....	45
Figura 34 - Código de inserção do usuário	46
Figura 35 - Código de exibição de dados minha conta	47
Figura 36 - Página minha conta	47
Figura 37 - Script malicioso cadastro do usuário.....	48
Figura 38 - Página minha conta com script malicioso	48
Figura 39 - Código de inserção do usuário com filtro	50
Figura 40 - Código de exibição de dados minha conta com filtro	50
Figura 41 - Script malicioso no cadastro do usuário após aplicação dos filtros.....	51
Figura 42 - Página minha conta com filtro aplicado.....	52
Figura 43 - Página de detalhes do produto	53
Figura 44 - Código de geração da quantidade de estrelas.....	53
Figura 45 - Script malicioso na URL da página de detalhes do produto.....	54
Figura 46 - Página de detalhes do produto com script malicioso	54
Figura 47 - Código injetado por meio do XSS DOM.....	55
Figura 48 - Código de geração da quantidade de estrelas com filtro	56
Figura 49 - Página de detalhes do produto com filtro aplicado	56

LISTA DE TABELAS

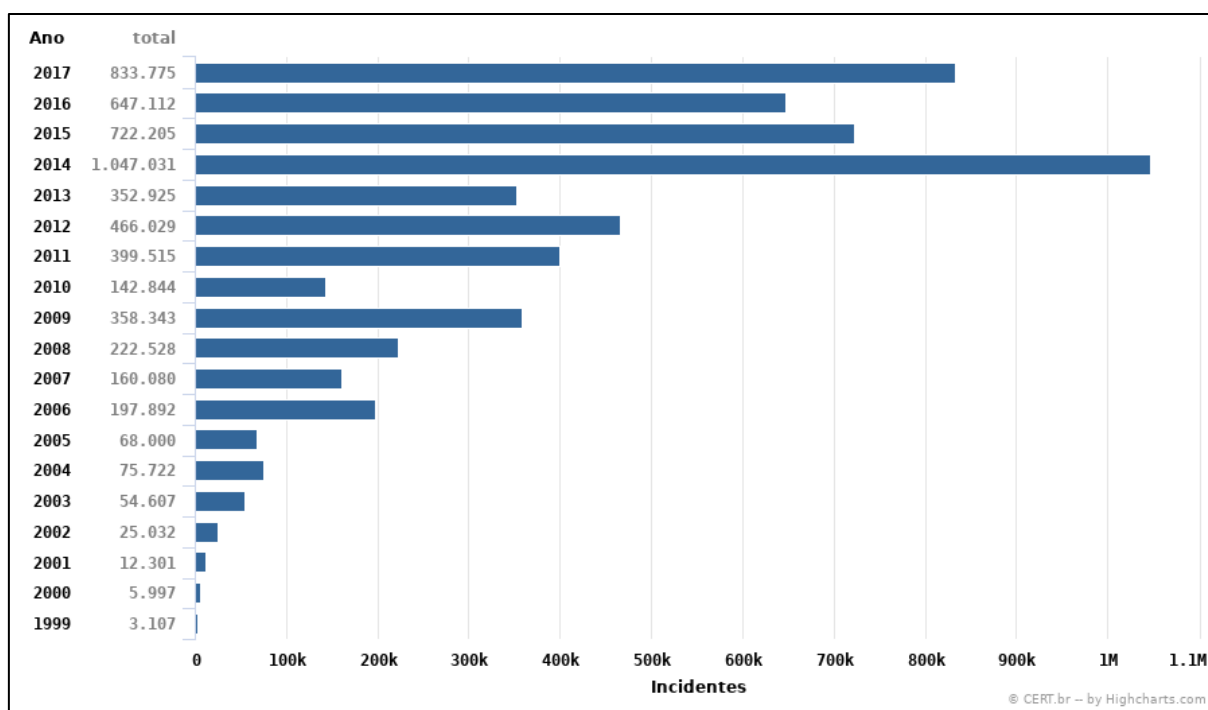
Tabela 1 - Caracteres especiais utilizando codificação entidade	35
Tabela 2 - Caracteres especiais utilizando codificação decimal.....	36
Tabela 3 - Caracteres especiais utilizando codificação hexadecimal	37
Tabela 4 - Caracteres especiais utilizando codificação percentual	38

1 INTRODUÇÃO

Com o advento da Internet e a utilização dos computadores em larga escala e dos dispositivos cada vez mais interconectados, o risco de vazamento de informações vem aumentando gradativamente ano após ano através dos ataques cibernéticos.

O Cert.BR é um grupo de resposta a incidentes da segurança da informação para a Internet do Brasil. Ele mantém todos os anos um gráfico de incidentes reportados pelos usuários da rede, como pode-se observar na figura 1.

Figura 1 - Total de incidentes reportados ao CERT.br por ano



Fonte: CERT.br (2017)

No gráfico mostrado na figura 1 é possível observar que no ano de 1999 o número de incidentes relacionados à segurança da informação era significativamente pequeno. Isto é fácil de entender devido ao cenário daquele ano, em que a Internet estava começando a se popularizar e grande parte das casas ainda não possuía conexão com a Internet.

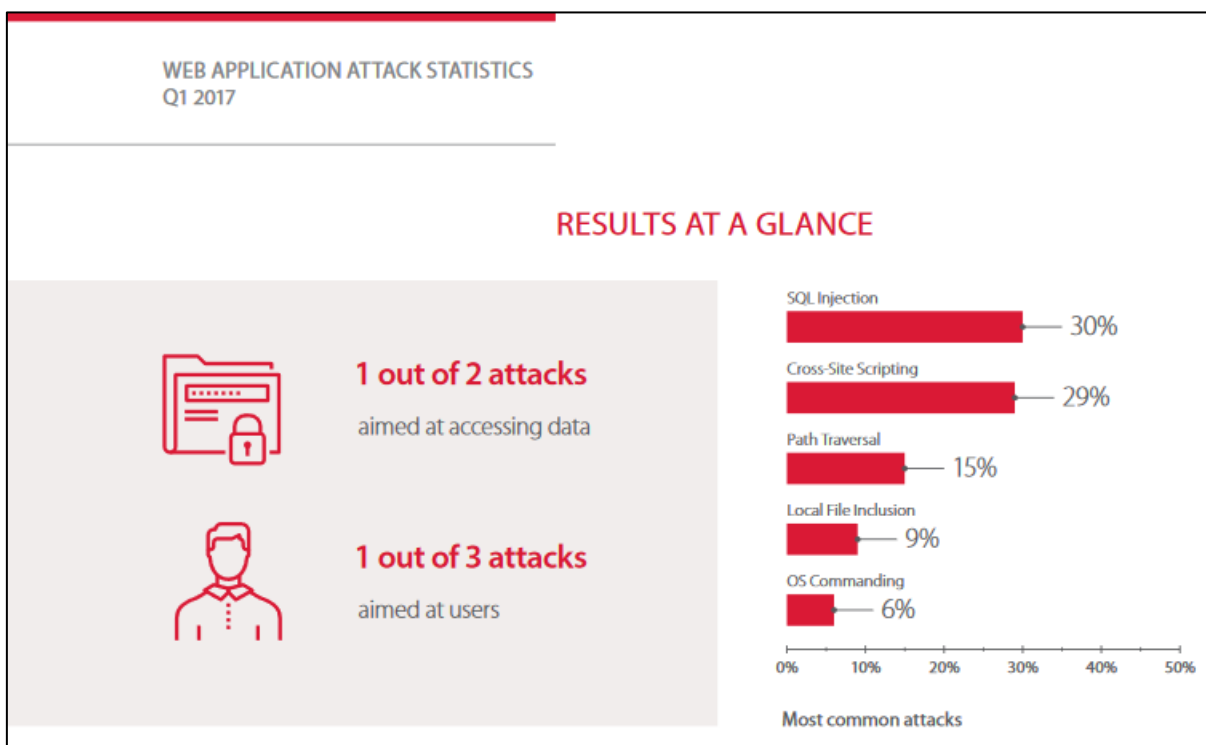
Porém conforme os anos foram se passando, a população foi obtendo acesso à rede mundial de computadores e o cenário foi se tornando cada vez pior em se tratando da segurança na rede. Vemos que um dos piores anos foi o ano de 2014,

os próximos anos tiveram uma baixa, mas logo subiram novamente. No quadro geral, a falta de segurança cibernética ainda é um problema para a população.

Em 2017 a empresa POSITIVE TECHNOLOGIES realizou um estudo dos ataques realizados em aplicações web reportados durante o primeiro quarto do mesmo ano. Tendo como prioridade a determinação dos tipos mais comuns de ataques, objetivos, intensidade e tempo de distribuição de ataques, além de analisar os ataques encontrados com mais frequência pelos clientes da empresa em diferentes setores. (PTSECURITY, 2017, p. 1).

Um dos principais ataques é o XSS (*Cross Site Scripting*), como pode-se observar na figura 2.

Figura 2 - Estatística da ocorrência do XSS 2017



Fonte: PTSECURITY (2017)

No gráfico mostrado na figura 2, visualiza-se que o ataque de XSS fica em segundo lugar com 29% de presença nos ataques realizados, só perde lugar para o *SQL Injection* com 30% de ocorrência no período.

Um bom exemplo ocorreu em outubro de 2005, onde a rede social MySpace sofreu um ataque de XSS conhecido como “*samy is my hero*”. Samy Kamkar, um usuário da rede que não estava contente com sua rede de amigos resolveu então

criar um script malicioso para aumentar sua rede de contatos. O script que Samy desenvolveu consistia em adicionar qualquer pessoa que visitasse sua página à sua lista de amigos e adicionava no perfil do infectado a frase “*samy is my hero*”. Além disso, o script se instalava no perfil da pessoa atacada possibilitando que mais pessoas fossem infectadas.

Grossman *et al.* (2007, p. 386) descreve o ocorrido:

“[...] Samy Kamkar lançou o maior *worm* da história da Internet. O que começou como uma brincadeira inocente, rapidamente se transformou em um enorme problema que fez com que os maiores sites de redes sociais ficassem off-line. Samy, junto com o resto da comunidade do *MySpace*, aprendeu uma valiosa lição naquele dia: As vulnerabilidades do XSS têm um poder além do que a maioria das pessoas entende. Concedido, o que Samy realizou não deve ser tido em alta estima, mas, ao mesmo tempo, este caso fez mais para aumentar a conscientização quanto aos perigos de aplicativos da Web inseguros [...]”

No ano de 2008 foi a vez da grande rede bancos Bradesco, que por uma falha de segurança em seu site permitiu um ataque XSS. De acordo com o site Linha Defensiva (2008):

“Criminosos brasileiros conseguiram descobrir um problema em uma página do Bradesco que permitia que a mesma fosse “alterada” por meio de links, possibilitando o uso do domínio do banco para todo tipo de atividade maliciosa. [...] crackers enviaram e-mail em massa contendo um link que explorava uma falha de XSS (Cross Site Scripting) existente em uma página localizada em institucional.bradesco.com.br. Se clicado, o link enviava informações à página que causavam um comportamento indesejável [...]”

Em 2010 a antiga rede social Orkut também foi alvo de um ataque XSS onde cerca de 600 mil perfis foram infectados pela falha. (G1 GLOBO, 2010).

São muitos os exemplos de ataques XSS bem-sucedidos realizados por todo o mundo, utilizando-se de técnicas maliciosas para ganhos pessoais, quebra de privacidade e até roubos de informação.

O objetivo desta obra é fornecer o conhecimento necessário sobre o ataque XSS tanto para o profissional de segurança da informação quanto para o público em geral. Também é objetivo deste trabalho propor e apresentar uma aplicação web vulnerável e a demonstração do ataque, bem como as formas de prevenção que podem ser aplicadas a fim de mitigar os riscos de ele vir a acontecer.

O presente trabalho foi estruturado em sete capítulos, sendo o primeiro a introdução da obra. Já no segundo capítulo serão abordados conceitos-chave para o entendimento do trabalho como o conceito de segurança da informação, bem como conceitos de aplicação *web*, as principais linguagens utilizadas no desenvolvimento web e qual a importância da linguagem de programação *Javascript* para o XSS.

No terceiro capítulo será abordado o conceito de XSS, como ele funciona e de que forma ele pode ser prejudicial a vida do usuário final. Também serão abordados os tipos de ataque XSS de acordo com sua classificação, nele o leitor será apresentado ao XSS refletido, o XSS armazenado e o XSS baseado em DOM (*Document Object Model*).

O quarto capítulo fornece uma visão dos tipos de codificação existentes, mostrando sua importância em especial nas aplicações *web* e como os atacantes se utilizam delas para realizar o ataque XSS.

O quinto capítulo apresenta uma aplicação web vulnerável ao XSS desenvolvida pelo autor, descrevendo os procedimentos de teste para a demonstração das vulnerabilidades e os resultados obtidos a partir deles.

O sexto capítulo dá ao leitor as ferramentas necessárias para a prevenção dos ataques de XSS a fim de mitigar os riscos envolvidos. O capítulo aborda a importância do cuidado ao inserir dados sensíveis do usuário numa aplicação web, bem como os procedimentos que podem ser executados pelo desenvolvedor de aplicações a fim de prevenir o ataque XSS.

No sétimo capítulo serão apresentadas as considerações finais e os resultados obtidos com este trabalho.

2 CONCEITOS INTRODUTÓRIOS

Neste capítulo serão abordados conceitos importantes pertinentes a área de estudo deste trabalho.

2.1 Segurança da Informação

Atualmente é possível observar uma crescente utilização de computadores e dispositivos móveis. A utilização destes dispositivos trouxe grande melhoria nos processos de comunicação, que facilitaram a troca de informações entre as pessoas e organizações. A informação é um ativo muito importante tanto para as pessoas como para as organizações.

Para Sêmola (2014, p.1):

"[...] é inegável que todas as empresas, independentemente de seu segmento de mercado, de seu *core business* e porte, em todas essas fases de existência, sempre usufruíram da informação, objetivando melhor produtividade, redução de custos, ganho de *market share*, aumento de agilidade, competitividade e apoio mais eficiente aos processos de tomada de decisão."

Dado o contexto apresentado é necessário a adoção de práticas voltadas para a preservação e disponibilidade das informações. Tais práticas são o objetivo da segurança da informação. A norma ABNT ISO/IEC 27002 (2005, p.x) define a segurança da informação como a proteção da informação de vários tipos de ameaças para garantir a continuidade do negócio, minimizar o risco ao negócio, maximizar o retorno sobre os investimentos e as oportunidades de negócio.

Toda esta preocupação é devida a crescente evolução da rede mundial de computadores, e em grande parte pelo aumento da utilização de aplicações web em tempo real. No próximo capítulo aprofundaremos este assunto.

2.2 Aplicações Web

Com o aumento da utilização da Internet, a utilização de aplicações web também se popularizou deixando de ser apenas utilizadas nos ambientes empresariais para serem amplamente adotadas pelo público em geral.

Para Winckler e Pimenta (2002, p.2):

"[..]. No início, a Web era apenas um ambiente para publicação de documentos no formato texto e HTML (*Hypertext Markup Language*)

e, portanto, a interação dos usuários era limitada a ler/imprimir texto e a selecionar links para outros documentos. Na sequência, vieram os formulários e programas CGIs (*Common Gateway Interface*) que permitem a entrada de dados do usuário e a integração com aplicações de banco de dados. Como consequência desta inovação, surgem aplicações complexas que utilizam a Web não apenas para troca de informações, mas como plataforma para aplicações distribuídas como, por exemplo, o comércio eletrônico e intranets (em que empresas usam a Web para gerenciar informações internas). ”

Quando um usuário utiliza qualquer dispositivo capaz de acessar a Internet para acessar mecanismos de busca, redes sociais, ou até mesmo um site específico, este usuário está acessando um serviço online hospedado em um servidor remoto que pode estar localizado em qualquer lugar do planeta. Este modelo é representado pelo que podemos chamar de modelo cliente/servidor. Este modelo propicia excelentes funcionalidades para o usuário, como recursos multimídia e audiovisuais, aplicações baseadas em banco de dados, sistemas distribuídos entre outros.

Todo este contexto trouxe melhoria para as aplicações e conseqüentemente trouxe benefícios para o usuário. No entanto devido a sua larga utilização, também surgiram alguns problemas como os ataques cibernéticos, incluindo o XSS que é o tema principal deste trabalho.

Para a realização dos ataques de injeção de código XSS são utilizadas algumas linguagens de programação para isto. O capítulo seguinte abordará as principais linguagens de programação da web e sua relação com o XSS.

2.3 As Linguagens da Web

A grande maioria das páginas web são constituídas de elementos que utilizam três linguagens principais, sendo elas o HTML (*HiperText Markup Language*), o CSS (*Cascading Style Sheets*) e o Javascript. A representação das três camadas de uma aplicação web pode ser observada na figura 3.

Figura 3 - Camadas de uma aplicação web



Fonte: Mozilla (2017)

O HTML é a linguagem de marcação utilizada para a exibição de conteúdo na página. Segundo Costa e Todeschini (2006, p.3) o HTML é uma linguagem de script muito utilizada no mundo da computação para web. Ela usa marcações que são usadas por navegadores web para interpretar o código descrito.

O CSS é a linguagem utilizada para estilizar os documentos HTML, sendo possível com ela alterar a cor dos elementos da página, tamanho ou até mesmo a fonte dos elementos

O Javascript é uma linguagem de programação que permite a implementação de itens complexos na página deixando-os dinâmicos, sendo possível utilizar funções que podem ser incluídas dentro do documento HTML.

Com Javascript você tem muitas possibilidades para “incrementar” seu documento HTML com elementos interessantes. Por exemplo, você será capaz de responder facilmente a eventos iniciados pelo usuário. (COSTA; TODESCHINI, 2006, p.55).

Das linguagens mencionadas anteriormente, o Javascript é a de maior relevância para este trabalho. A capacidade de um usuário mal-intencionado em

executar Javascript arbitrário no navegador de outro usuário é extremamente prejudicial. Para Kallin e Valbuena (2013) o invasor pode através da linguagem Javascript executar os seguintes tipos de ataque:

“Roubo de cookies: O invasor pode acessar os cookies da vítima associados ao site usando document.cookie, enviá-los para seu próprio servidor e usá-los para extrair informações confidenciais, como identificações de sessão.

Keylogging: O invasor pode registrar um ouvinte de evento de teclado usando addEventListener e, em seguida, enviar todas as combinações de teclas do usuário para seu próprio servidor, possivelmente registrando informações confidenciais, como senhas e números de cartão de crédito.

Phishing: O invasor pode inserir um formulário de login falso na página usando a manipulação de DOM, definir o atributo de ação do formulário para direcionar seu próprio servidor e depois enganar o usuário para enviar informações confidenciais. ”

3 O CROSS SITE SCRIPTING

O *Cross Site Scripting (XSS)* é um tipo de ataque cibernético comum a sites e aplicações web. Resumidamente, o ataque consiste na injeção de código malicioso na aplicação através de *inputs* não validados pelo desenvolvedor da aplicação.

De acordo com a Owasp (2018):

“Os ataques de Cross-Site Scripting (XSS) são um tipo de injeção, na qual scripts maliciosos são injetados em sites de outra forma benignos e confiáveis. Ataques XSS ocorrem quando um invasor usa um aplicativo da Web para enviar código mal-intencionado, geralmente na forma de um script do lado do navegador, para um usuário final diferente. As falhas que permitem que esses ataques sejam bem-sucedidos são bastante difundidas e ocorrem em qualquer lugar em que um aplicativo da Web use a entrada de um usuário na saída gerada sem validá-lo ou codificá-lo.”

Os ataques de *Cross Site Scripting* ocorrem em sua maioria quando existe a entrada de dados em uma aplicação web através de um código-fonte que não possui validação da entrada de dados do usuário. Esses dados maliciosos são incluídos no conteúdo dinâmico que é enviado ao servidor web sem ser validado. As imagens a seguir demonstram exemplos de código potencialmente suscetíveis ao ataque XSS:

Figura 4 - Código inseguro que escreve tags no documento HTML

```
document.write("<p>texto<p>");  
document.writeln("<script>console.log('isto pode ser perigoso!');");
```

Fonte: Elaborado pelo autor

Figura 5 - Código inseguro que adiciona um conteúdo em um elemento HTML

```
var elemento = document.getElementById("meu-elemento");  
elemento.innerHTML = "<script>alert('Perigo!');</script>";
```

Fonte: Elaborado pelo autor

O código-fonte apresentado na figura 4 escreve um texto ou um conjunto de *tags* HTML no documento sem realizar nenhum tipo de filtragem de dados. Um criminoso cibernético poderia então se aproveitar do código inseguro para tentar inserir um script malicioso que será interpretado pela página e executado, podendo assim causar danos ao usuário. A figura 5 contém um código-fonte potencialmente

perigoso também, onde é adicionado um conteúdo texto dentro de um elemento HTML da mesma forma que no exemplo anterior sem qualquer filtragem.

Os exemplos mencionados anteriormente contêm código malicioso injetado através da *tag* `script` do HTML, porém os ataques podem facilmente ser realizados sem o uso das *tags* `<script></script>`. Alguns tipos de *tag* farão exatamente o mesmo efeito, como se pode ver nos exemplos a seguir:

A seguir, verifica-se um script malicioso que será executado após o término do carregamento da página, podendo comprometer a navegação e até mesmo os dados do usuário que esteja utilizando a aplicação, como pode-se observar na figura 6.

Figura 6 - Ataque XSS utilizando o atributo `onload` da tag `<body>` do HTML

```
<body onload=alert('hackedo!')>
```

Fonte: Elaborado pelo autor

Na figura 7 veremos um exemplo de ataque XSS utilizando os eventos *onmouseover* e *onerror* das *tags* HTML, que podem ser bastante exploradas por um usuário mal-intencionado.

Figura 7 - Ataque XSS utilizando os atributos `onmouseover` e `onerror`

```
<strong onmouseover=alert('Ops!')>  

```

Fonte: Elaborado pelo autor

Na imagem anterior podemos ver que na *tag* `` foi adicionado um script malicioso no evento *onmouseover*, onde ao passar o mouse pelo elemento o usuário seria infectado. No outro exemplo vemos que o atributo *src* *tag* `` foi infectado com uma URL (*Uniform Resource Locator*) inexistente e também foi adicionado um script malicioso no evento *onerror* da mesma *tag*, causando o disparo do script no momento em que o navegador não conseguir carregar a URL. Em todos os casos citados, o usuário seria afetado de forma bastante negativa.

O XSS pode causar uma variedade de problemas para o usuário final que variam em gravidade, desde um aborrecimento até o comprometimento total da conta. (OWASP, 2018).

No pior dos casos, um ataque XSS pode causar o sequestro da sessão do usuário, fazendo assim com que um outro usuário mal-intencionado assuma aquela conta. Um usuário mal-intencionado também pode ter acesso a arquivos sigilosos e causar o vazamento de informações, plantar vírus na página alvo, além de poder modificar o conteúdo da página ou até mesmo redirecionar o usuário para outro site. Este último exemplo é extremamente perigoso, pois, o usuário final pode por exemplo ser levado a acessar uma página de banco fictícia com a mesma aparência de seu banco, solicitando então a entrada de seus dados bancários como número de cartão de crédito e senha.

3.1 CLASSIFICAÇÃO QUANTO AOS TIPOS DE XSS

Existem três principais categorias de XSS, que são normalmente direcionadas ao navegador do usuário. Elas são o XSS refletido, o XSS armazenado e o XSS baseado em DOM.

A Owasp (2017) categoriza os três tipos de XSS de acordo com o ambiente afetado por eles conforme a figura 8.

Figura 8 - Tipos de XSS de acordo com ambiente afetado

Tipo de XSS	Servidor	Cliente (Usuário)
Armazenado (Stored)	É armazenado no servidor	É armazenado no cliente
Refletido (Reflected)	É refletido no servidor	É refletido no cliente
Baseado em DOM (DOM Based)	Sem efeito	É executado no cliente

Fonte: Owasp (2017)

Na tabela mostrada na figura 8, pode-se observar que o tipo XSS Armazenado e o XSS Refletido se comportam tanto no servidor web quanto no cliente que está executando a aplicação. No entanto o XSS baseado em DOM trabalha apenas no lado do cliente.

Também é importante notar o nível de criticidade de cada um dos tipos de XSS, conforme pode ser observado na figura 9.

Figura 9 - Tipos de XSS de acordo com o nível de criticidade

Tipo de XSS	Nível de criticidade
Armazenado (Stored)	Alto
Refletido (Reflected)	Médio
Baseado em DOM (DOM Based)	Médio

Fonte: Elaborado pelo autor

Nas próximas seções serão abordados cada tipo de XSS em profundidade.

3.1.1 XSS Armazenado

No ataque de XSS armazenado ou persistido um usuário mal-intencionado injeta um script malicioso que pode ser permanentemente armazenado no servidor web da aplicação. Neste caso, o usuário se torna uma vítima quando tenta utilizar uma área do sistema ou uma página contendo o código malicioso armazenado.

De acordo com a Owasp (2018) os ataques XSS armazenados:

“[...] são aqueles em que o script injetado é armazenado permanentemente nos servidores de destino, como em um banco de dados, em um fórum de mensagens, registro de visitantes, campo de comentários etc. A vítima recupera o script mal-intencionado do servidor quando solicita uma informação armazenada. O XSS armazenado também é conhecido como XSS Persistente ou Tipo-I.”

Esses tipos de XSS são geralmente mais significativos do que outros, uma vez que um usuário mal-intencionado pode potencialmente atingir um grande número de usuários apenas com uma ação específica e facilitar o processo de engenharia social. (REDE SEGURA, 2012).

Uto (2013, p.182), define os passos para realizar um XSS armazenado:

1. Um usuário malicioso insere código no corpo da pergunta e a submete à aplicação defeituosa, que, por esse motivo não valida a entrada e armazena o texto integralmente no banco de dados.
2. Outro usuário verifica a lista de perguntas e resolve acessar justamente aquela com conteúdo malicioso.

3. A aplicação recupera a informação do banco de dados e a utiliza para gerar a página solicitada, sem codificar a saída.
4. O Javascript escrito pelo atacante é transportado até o navegador do usuário, onde é executado, efetuando atividades maliciosas.

Segundo Owasp (2018), o XSS armazenado é considerado um risco de nível alto a crítico já que isto é extremamente perigoso em sistemas como CMS (*Content Management System*), blogs ou fóruns de discussão, onde uma grande quantidade de usuários acessará entradas de outros usuários.

O exemplo a seguir, exibe um formulário da web para cadastro de usuário e senha hipoteticamente sem validação de dados de entrada. No exemplo, um usuário mal-intencionado pode inserir um script malicioso em qualquer um dos campos, que então será armazenado no banco de dados, como pode-se observar na figura 10.

Figura 10 - Formulário sem validação XSS armazenado



Fonte: Elaborado pelo autor

No exemplo apresentado na figura 10, um usuário mal-intencionado propositalmente adiciona um script malicioso em um dos campos do formulário a fim de infectar o sistema. O resultado disso é que a informação fornecida é salva em um banco de dados do sistema, assim quando a aplicação recuperar o dado contido banco de dados, o que será executado na realidade é o script do atacante.

A figura 11 demonstra como a página afetada se comportaria ao receber o código infectado:

Figura 11 - Sistema infectado pelo XSS armazenado



Fonte: Elaborado pelo autor

3.1.2 XSS Refletido

O ataque de XSS refletido como o próprio nome diz, é aquele que reflete as ações do usuário no servidor web. Eles podem ser refletidos em um resultado de uma busca de um site ou até mesmo uma mensagem de alerta ou erro.

De acordo com a Owasp (2018):

“Os ataques refletidos são entregues às vítimas por meio de outra rota, como em uma mensagem de e-mail ou em outro site. Quando um usuário é levado a clicar em um link malicioso, enviar um formulário especialmente criado ou até mesmo navegar em um site malicioso, o código injetado viaja para o site vulnerável, que reflete o ataque de volta ao navegador do usuário. O navegador então executa o código porque veio de um servidor "confiável". O XSS refletido também é conhecido como XSS Não Persistente ou Tipo II.”

Uto (2013, p.181), define em linhas gerais, os passos gerais para que a exploração aconteça:

1. O atacante fornece à vítima um link para a aplicação vulnerável, com código Javascript embutido em um dos parâmetros da URL.
2. A vítima solicita à aplicação vulnerável o recurso identificado pela URL fornecida no primeiro passo deste roteiro.
3. A aplicação atende a requisição e reflete o código malicioso para o navegador do usuário.

4. O Javascript escrito pelo atacante é executado na máquina do usuário, como se fosse proveniente da aplicação, e, portanto, com todos os privilégios de um script legítimo.

A seguir veremos um código na linguagem PHP (*PHP Hypertext Preprocessor*) que é potencialmente suscetível ao ataque XSS refletido conforme a figura 12.

Figura 12 - Código vulnerável ao XSS refletido

```
<?php
$consulta = $_GET["busca"];
// ... Executa uma consulta SQL qualquer
echo "<div>"
    + "Não foram encontrados registros para a busca: " . $consulta . "!";
    + "</div>";
?>
```

Fonte: Elaborado pelo autor

No código-fonte apresentado acima é possível visualizar que a variável “busca” recebida como parâmetro HTTP (*Hyper Text Transfer Protocol*) da URL não possui qualquer tipo de filtragem, e logo em seguida ela é devolvida ao navegador web do usuário.

Se for realizado uma análise desta requisição, perceberemos que a URL processa uma requisição do tipo GET, que consiste em passar parâmetros diretamente na *query string* formatada. Sendo assim, pode-se obter como entrada do usuário o texto conforme pode ser visto na figura 13.

Figura 13 - Exemplo de busca com parâmetro pela URL

```
?busca=Busca+Do+Usuario
```

Fonte: Elaborado pelo autor

O parâmetro informado na URL poderia ser facilmente alterado por um usuário mal-intencionado para o conteúdo mostrado na figura 14, onde o script malicioso seria executado diretamente na página web podendo causar danos para o usuário:

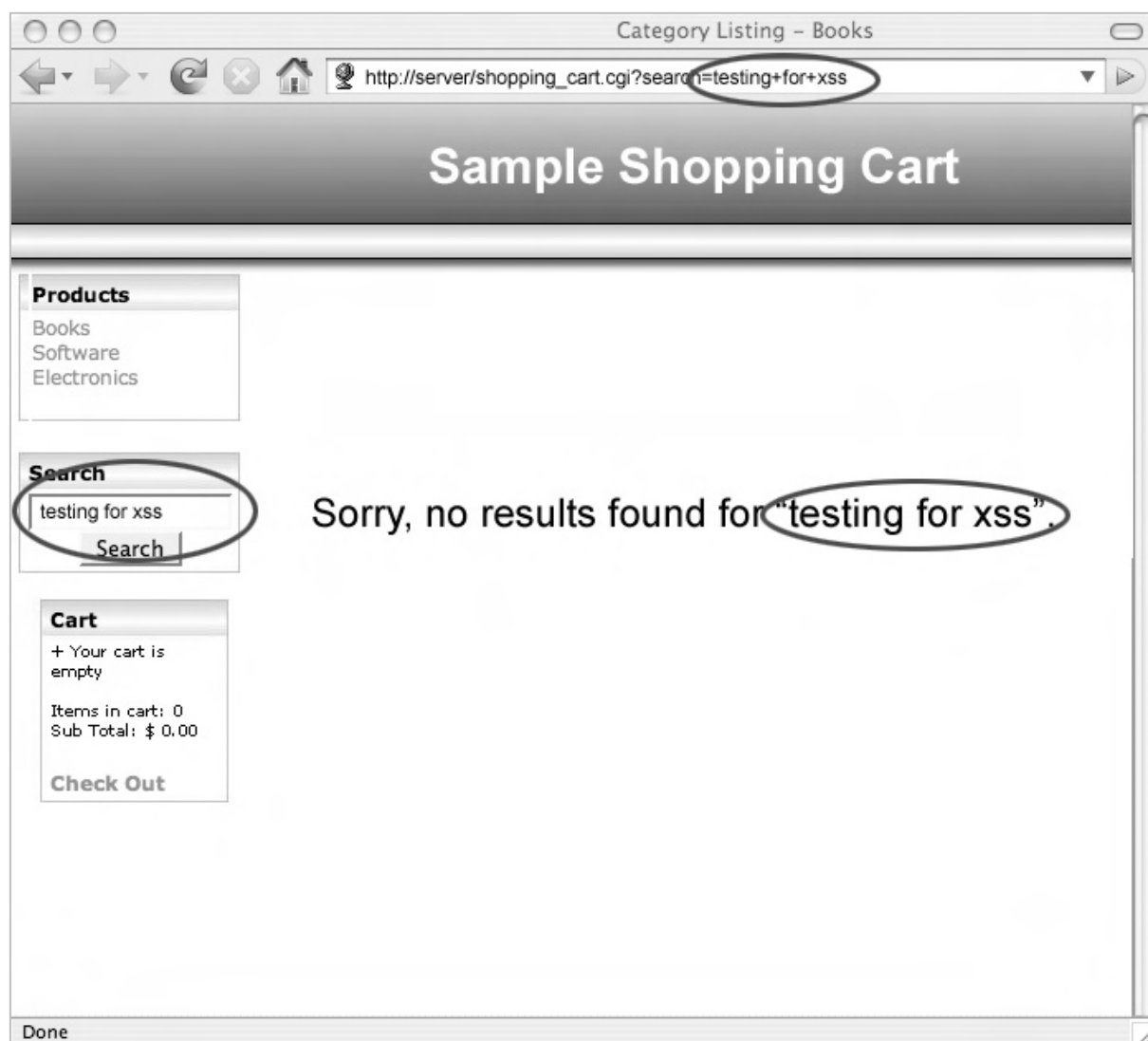
Figura 14 - Exemplo de busca com script malicioso

```
?busca=<script>alert('Ataque XSS!');</script>
```

Fonte: Elaborado pelo autor

No próximo exemplo será apresentada uma página web de uma loja virtual onde possui um formulário de busca de produtos sem qualquer filtragem dos dados de entrada do usuário. Ao informar um valor qualquer no campo de busca do formulário, ele será refletido para a página web, como pode ser observado na figura 15.

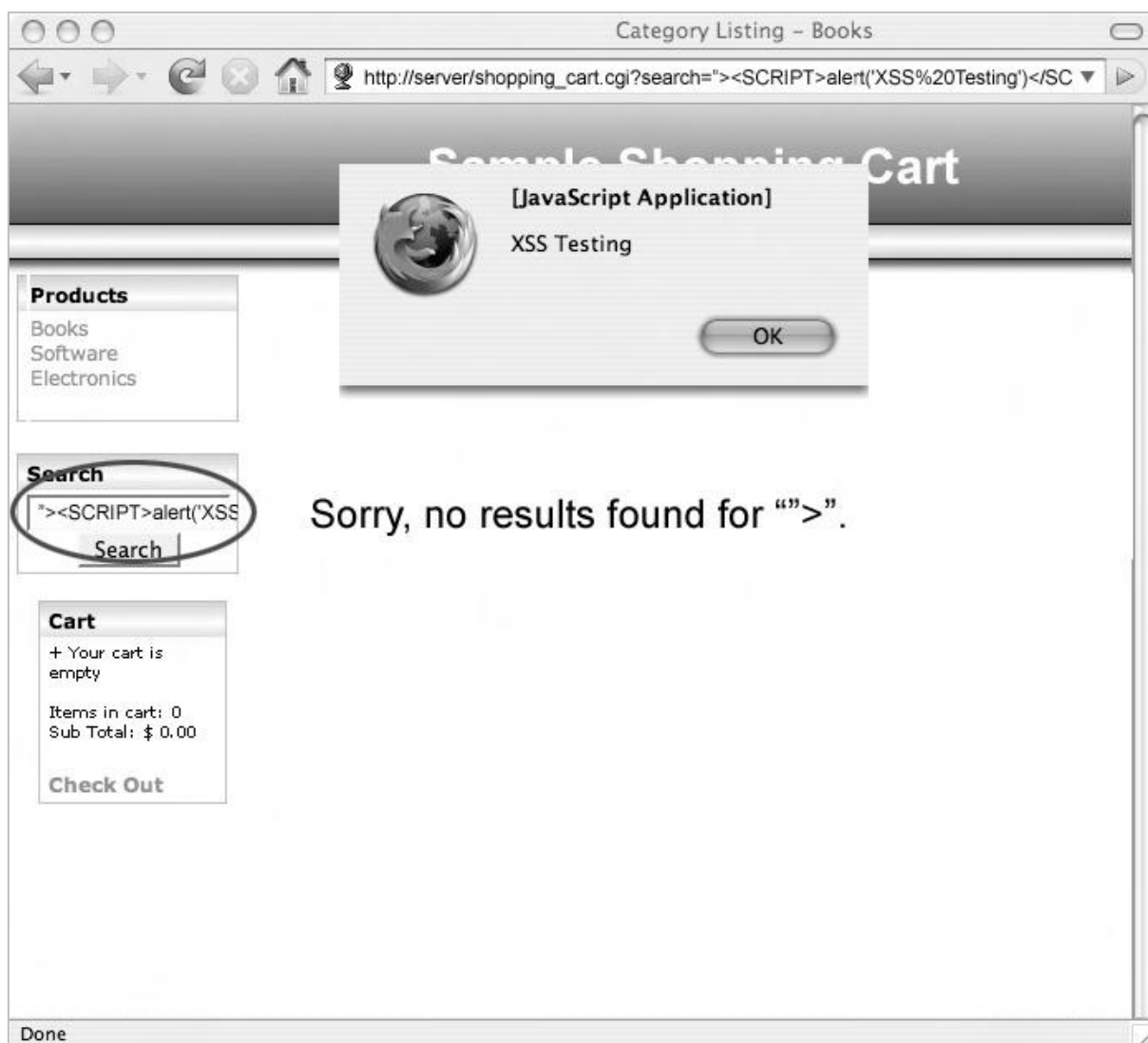
Figura 15 - Exemplo XSS refletido em uma loja virtual



Fonte: XSS Attacks Cross Site Scripting Exploits, p. 70, 2007

Ao informar o script "><SCRIPT>alert('XSS%20Testing')</SCRIPT>" no campo de busca da página web, o dado informado é imediatamente refletido na página de resultados da busca, podendo assim, causar danos ao usuário do sistema, como pode ser observado na figura 16.

Figura 16 - Exemplo de página infectada com XSS refletido em uma loja virtual



Fonte: XSS Attacks Cross Site Scripting Exploits p. 71, 2007

3.1.3 XSS Baseado em DOM

Para entender o que é o ataque do XSS baseado em DOM, é necessário primeiro conhecer o que é DOM. De acordo com Mozilla (2017):

“O Modelo de Objeto de Documento (DOM) é uma interface de programação para documentos HTML, XML e SVG. Ele fornece uma representação estruturada do documento como uma árvore. O DOM define métodos que permitem acesso à árvore, para que eles

possam alterar a estrutura, estilo e conteúdo do documento. O DOM fornece uma representação do documento como um grupo estruturado de nós e objetos, possuindo várias propriedades e métodos. [..]”

O DOM é a representação da árvore de elementos HTML contida em uma página web, através dele é possível manipular os elementos da página web com a linguagem de programação Javascript.

A seguir será apresentado um arquivo de código-fonte com os elementos básicos de um a página HTML, também foram adicionados alguns elementos no DOM para que se possa demonstrar a efetividade do Javascript na manipulação da árvore do DOM, conforme pode-se observar na figura 17.

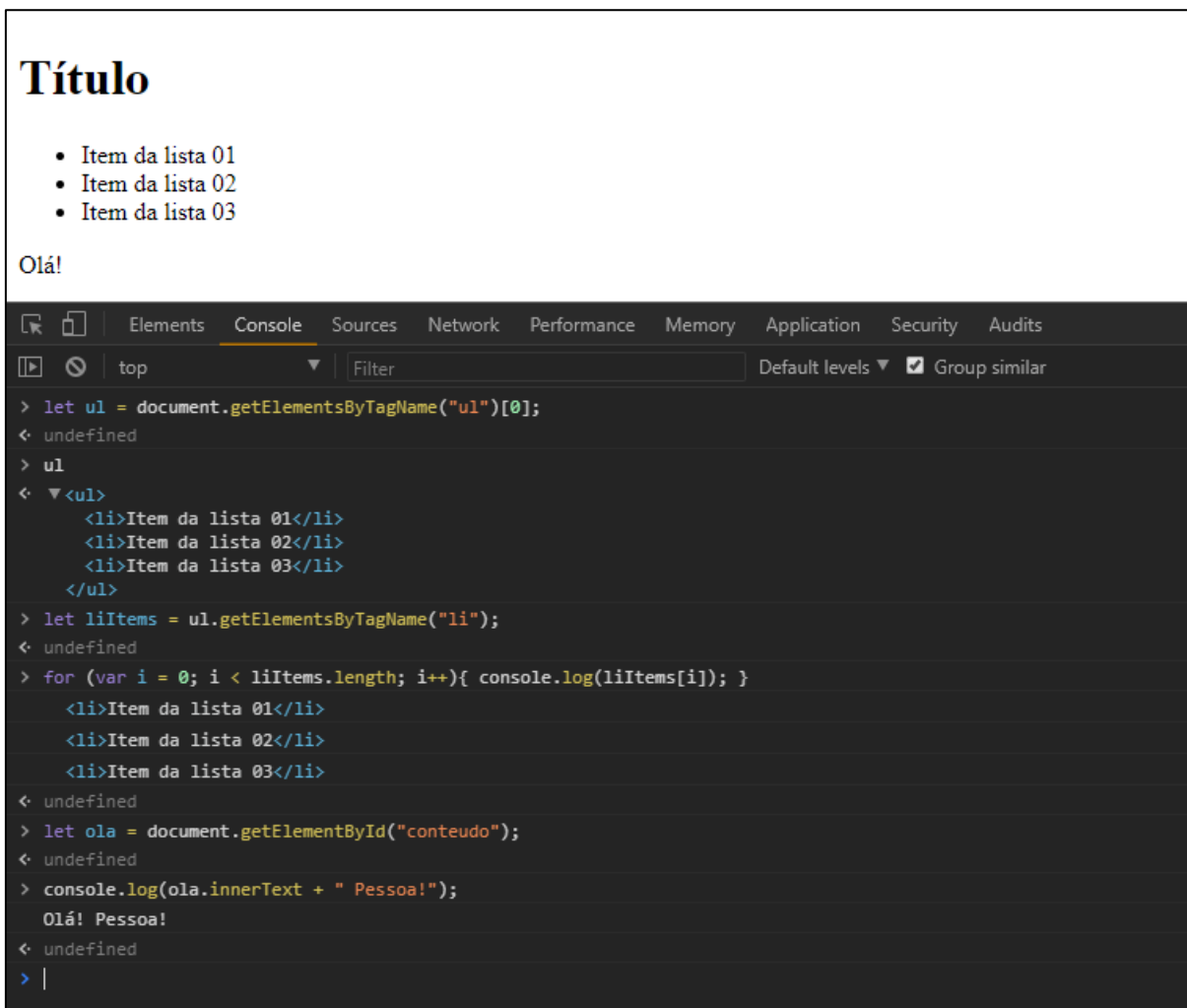
Figura 17 - Código HTML para demonstração do DOM

```
1 <!DOCTYPE html>
2 <html lang="en" dir="ltr">
3   <head>
4     <meta charset="utf-8">
5     <title>Sistema Web 1.0</title>
6   </head>
7   <body>
8     <h1>Título</h1>
9     <ul>
10      <li>Item da lista 01</li>
11      <li>Item da lista 02</li>
12      <li>Item da lista 03</li>
13    </ul>
14    <div id="conteudo">Olá!</div>
15  </body>
16 </html>
```

Fonte: Elaborado pelo autor

Na figura 18 serão demonstradas algumas manipulações que podem ser realizadas no DOM através do Javascript, utilizando apenas o console de desenvolvedor do navegador web. Pode-se verificar que é possível manipular os elementos de diversas formas, seja recuperando um elemento específico através de seu identificador único “id”, ou buscando um elemento pelo nome de sua *tag*, ou até mesmo percorrer elementos filhos de um determinado elemento-pai HTML.

Figura 18 - Manipulação de elementos HTML através do DOM



Fonte: Elaborado pelo autor

O ataque de XSS baseado em DOM é bastante similar ao XSS refletido, mas difere deste por não necessitar que o código malicioso seja enviado ao servidor. Consequentemente, o efeito precisa estar no lado cliente da aplicação. (UTO, 2013, p.182).

A seguir será apresentado um código de exemplo de XSS baseado em DOM. O exemplo consiste na exibição de uma mensagem personalizada de boas-vindas, onde o nome do usuário é obtido de um parâmetro da URL através da linguagem Javascript. Como pode ser observado na figura 19.

Figura 19 - Exemplo de código XSS baseado em DOM

```
<html>
<head><title>XSS baseado DOM</title></head>
<body>
<h2>
<script>
var url = document.URL;
var pos = url.indexOf("usuario=");
document.write('Bem-vindo');
if (pos != -1) {
    document.write(', ' + unescape(url.substring(pos + 8, url.
length)));
}
document.write('!!');
</script>
</h2>
</body>
</html>
```

Fonte: Teste de Invasão de Aplicações Web, p.183, 2013

No cenário da figura 19, o atacante pode enviar a página HTML com este script que verifica o parâmetro “usuário” e caso esteja presente, o valor dele é escrito no documento por meio do método *document.write()* que manipula o DOM da página web.

Neste caso específico de exemplo, o script em si não causa mal algum para o usuário, no entanto numa ameaça real o script poderia conter código malicioso, ou até mesmo poderia realizar um redirecionamento para uma página do atacante onde poderia ser realizada a captura de dados sensíveis do usuário, como dados de cartão de crédito, por exemplo.

Uto (2013, p.184), define os passos gerais para que o XSS baseado em DOM aconteça:

1. O atacante fornece à vítima um link para a aplicação vulnerável, com código Javascript embutido como um fragmento de URL.
2. A vítima solicita à aplicação vulnerável o recurso identificado pela URL fornecida no primeiro passo deste roteiro. Note que o fragmento não é enviado ao servidor junto com a requisição.
3. A aplicação fornece como resposta à requisição uma página HTML contendo código Javascript, o qual altera o documento, com base no valor de um parâmetro da URL. Nesse momento, o código malicioso é inserido no DOM, juntamente com o argumento especificado.
4. O Javascript escrito pelo atacante é, então, executado na máquina do usuário, como se fosse proveniente da aplicação, e, portanto, com todos os privilégios de um script legítimo.

4 CODIFICAÇÕES

A linguagem HTML possui caracteres que são considerados especiais, estes não são exibidos corretamente no navegador web da mesma forma como foram inseridos no código-fonte, portanto para que seja possível realizar a exibição destes caracteres especiais é necessário um tipo de codificação própria que é chamado de padrão ASCII.

“ASCII (do inglês *American Standard Code for Information Interchange*; "Código Padrão Americano para o Intercâmbio de Informação") é um código binário (cadeias de bits: 0s e 1s) que codifica um conjunto de 128 sinais: 95 sinais gráficos (letras do alfabeto latino, sinais de pontuação e sinais matemáticos) e 33 sinais de controle, utilizando, portanto, apenas 7 bits para representar todos os seus símbolos” (CONHECIMENTO GERAL, 2016)

As codificações são de extrema importância no contexto dos ataques XSS. Nos próximos capítulos serão apresentadas as principais formas de codificação utilizadas no desenvolvimento de páginas web.

4.1 Codificação HTML

De acordo com Costa e Todeschini (2006, p.13) existem basicamente duas nomenclaturas impostas para o uso de caracteres especiais na linguagem HTML:

“1 – A nomenclatura da entidade HTML: Para representar um caractere especial a partir desta nomenclatura, inicia-se a representação com o caractere ‘&’, em seguida e usado um nome que identificará o caractere terminado com o caractere ‘;’

2 – A nomenclatura para a norma ISO Latin-1: Usa a tabela ASCII como referência. Para representar um caractere especial a partir da norma ISO Latin-1, inicia-se a representação com os caracteres “&#”, em seguida, insira a posição da tabela ASCII que representa o caractere desejado, por fim, termine com o caractere ‘;’.

A seguir será abordada a codificação entidade que utiliza a nomenclatura da entidade HTML, representando os caracteres acentuados, pontuações e marcadores. Também serão abordadas as codificações em decimal e hexadecimal que possuem o mesmo efeito dos caracteres especiais e se utilizam da nomenclatura ISO Latin-1.

4.1.1 Codificação entidade

A codificação entidade é bastante utilizada pelos atacantes durante a prática do XSS. Neste tipo de codificação, o caractere a ser codificado é representado por `&<nome da entidade>;`;

Uto (2013, p.21) entende que:

“[...] os caracteres especiais possuem nomes específicos em HTML, os quais podem ser empregados no processo de codificação. Por exemplo, o caractere “<” é codificado como “<”, pois “lt” é o nome de entidade especificado pelo padrão para esse símbolo.”

A seguir serão apresentados alguns dos caracteres que fazem uso da codificação entidade, como pode ser observado na tabela 1.

Tabela 1 - Caracteres especiais utilizando codificação entidade

Caractere reservado	Caractere codificado
Espaço	
“	"
‘	'
&	&
<	<
>	>

Fonte: Faria (2011)

Em um ataque XSS, o atacante utiliza de diversas técnicas para explorar a aplicação. Dentre elas a injeção de código malicioso pelo DOM é bastante utilizada.

No exemplo a seguir, o atacante pode vir a injetar um elemento HTML na página durante o carregamento da *tag* ``. Caso não seja possível carregar, ele efetuará o disparo do evento *onerror*, o qual irá redirecionar o usuário para outro domínio, conforme pode ser observado na figura 20.

Figura 20 - Ataque XSS sem codificação

```

17
18         
20

```

Fonte: Elaborado pelo autor

Contudo, os caracteres especiais como ‘<’ ou ‘>’ não estão codificados. Portanto podem haver filtros na aplicação que impossibilitem a injeção destes tipos de caracteres. Para que isso não ocorra, o atacante pode-se utilizar da codificação entidade. Assim, as chances de sucesso no ataque serão maiores, conforme pode ser observado na figura 21.

Figura 21 - Ataque XSS com codificação entidade

```

17
18     
20

```

Fonte: Elaborado pelo autor

4.1.2 Codificação decimal

A codificação decimal utiliza o valor Unicode do caractere em base decimal e segue o padrão `&#<número decimal>`. Por exemplo, o caractere "<" é codificado como "<".

A tabela 2 apresenta alguns dos caracteres que fazem uso da codificação decimal.

Tabela 2 - Caracteres especiais utilizando codificação decimal

Caractere reservado	Caractere codificado
Espaço	
“	"
‘	'
&	&
<	<
>	>

Fonte: Faria (2011)

Utilizando o exemplo da seção anterior, veremos adiante que o atacante pode-se utilizar da codificação decimal para obter sucesso em seu ataque, conforme pode ser observado na figura 22.

Figura 22 - Ataque XSS com codificação decimal

```

19
20     
22

```

Fonte: Elaborado pelo autor

4.1.3 Codificação hexadecimal

Na codificação hexadecimal também é utilizado o valor Unicode do caractere, no entanto ele é representado em base hexadecimal e utiliza a padrão `&#x<número hexadecimal>`. Aplicando o exemplo do caractere “<”, obtém-se “<”.

A tabela 3 apresenta alguns dos caracteres que fazem uso da codificação hexadecimal.

Tabela 3 - Caracteres especiais utilizando codificação hexadecimal

Caractere reservado	Caractere codificado
Espaço	
“	"
‘	'
&	&
<	<
>	>

Fonte: Faria (2011)

Na figura 23 é possível observar como ficaria o código utilizado pelo atacante nos exemplos anteriores aplicando a codificação hexadecimal.

Figura 23 - Ataque XSS com codificação hexadecimal

```

28
29     
31

```

Fonte: Elaborado pelo autor

4.2 Codificação percentual

A codificação percentual consiste na exibição de caracteres com significados especiais utilizados pelas URLs. Uto (2013, p.20) entende que:

“Uma URL – ou mais geralmente uma URI – pode conter somente caracteres ASCII imprimíveis, conforme especificado pela RFC 3986. Alguns deles possuem significado especial em URLs, atuando como delimitadores, e, assim, são classificados como reservados. Quando precisam ser utilizados como dados, nesse contexto, devem ser codificados, para que possam ser corretamente identificados como tais. O método empregado, chamado de codificação de URL ou codificação percentual, consiste no uso de um caractere “%” seguido de dois dígitos hexadecimais, que representam o valor numérico do dado sendo codificado.”

A tabela 4 apresenta alguns dos caracteres que fazem uso da codificação percentual comumente utilizados nas URLs.

Tabela 4 - Caracteres especiais utilizando codificação percentual

Caractere reservado	Caractere codificado
Espaço	%20
“	%22
‘	%27
#	%23
%	%25
&	%26
<	%3C
=	%3D
>	%3E
@	%40

Fonte: Faria (2011)

A figura 24 apresenta um arquivo de código-fonte com um formulário HTML e um script escrito na linguagem Javascript que consiste em obter o valor inserido pelo usuário, converter para a codificação percentual e apresentar o resultado na tela do navegador do usuário.

Figura 24 - Código para demonstração da codificação de URL

```

2  <form onsubmit="return false;">
3    <div>
4      <label for="texto">Texto:</label>
5      <input id="texto" type="text">
6      <button onclick="encodeURIComponent()">Encode URL</button>
7    </div>
8    <div id="resultado"></div>
9  </form>
10 <script>
11 function encodeURIComponent() {
12   var texto = document.getElementById("texto").value;
13   document.getElementById("resultado").innerHTML = encodeURI(texto);
14 }
15 </script>

```

Fonte: Elaborado pelo autor

O código-fonte apresentado na figura 24 tem como resultado a página web a seguir, como pode ser observado na figura 25.

Figura 25 - Exemplo de codificação percentual

Texto: <input type="text" value="entrada do usuário"/> <input type="button" value="Encode URL"/>
entrada%20do%20usu%C3%A1rio

Fonte: Elaborado pelo autor

A codificação em porcentagem é bastante explorada pelos atacantes que realizam o XSS. Uma boa parte dos servidores web modernos verifica a presença destes códigos e os bloqueiam de serem executados, no entanto existem alguns tipos de servidores que não os bloqueiam por padrão sendo necessário que este filtro seja realizado também pelo desenvolvedor da aplicação.

Na figura 26 o atacante injeta um código malicioso que redireciona o site para outro domínio utilizando a codificação percentual.

Figura 26 - Injeção de código na URL através de codificação percentual

```

34
35  http://site.com?id= window.location%3D%E2%80%9Dhttp%3A%2F%2Fwww.outro.com.br%
36

```

Fonte: Elaborado pelo autor

5 ANÁLISE DE VULNERABILIDADE DE UMA APLICAÇÃO WEB AO XSS

Neste capítulo será apresentada a aplicação Loja TCC, uma aplicação web desenvolvida pelo autor deste trabalho que consiste em um sistema que simula uma loja virtual onde há uma página de produtos, busca por produtos, login e área privativa do usuário onde os produtos cadastrados podem ser atualizados pelo usuário autenticado no sistema.

A programação do sistema foi realizada utilizando a linguagem de programação PHP e o banco de dados MySQL, também foram utilizadas as linguagens Javascript, HTML e CSS. A aplicação foi desenvolvida de modo que não haja nenhum tipo de filtros de dados, portanto será possível explorar cada um dos tipos de injeção utilizados pelo ataque de XSS.

5.1 Procedimentos de teste

Todos os testes foram realizados em ambiente controlado, utilizando uma aplicação construída pelo próprio autor.

Nas próximas seções serão abordadas a execução dos ataques de XSS refletido, XSS armazenado e XSS baseado em DOM na aplicação desenvolvida pelo autor do presente trabalho.

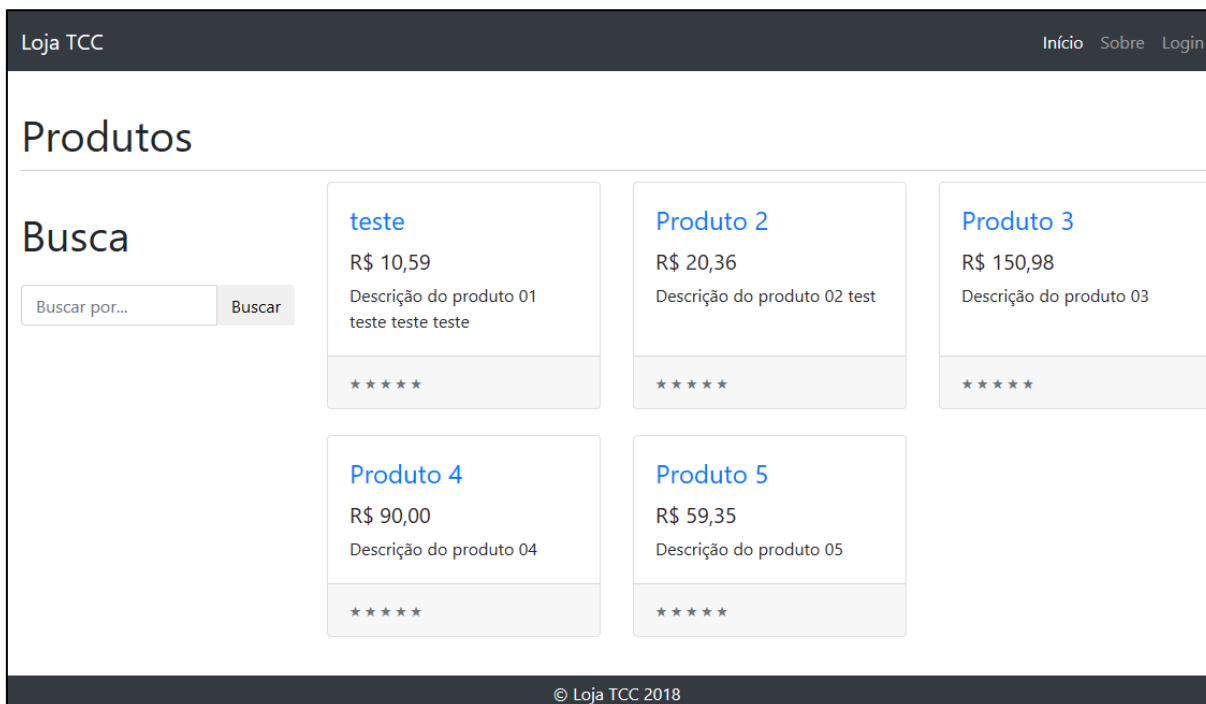
5.2 Execução do ataque XSS refletido

Como pôde ser visto nos capítulos anteriores, o XSS refletido consiste na injeção de scripts maliciosos em aplicações web, estes são refletidos em páginas que não possuem validação de filtros de entrada de dados. Na próxima seção serão abordados os procedimentos de exploração do XSS refletido na aplicação desenvolvida pelo autor, bem como as formas de prevenção da mesma.

5.2.1 Explorando a vulnerabilidade

Para exemplificar o XSS refletido dentro da aplicação desenvolvida será utilizado a página de produtos a qual contém um formulário com um campo de busca, conforme pode ser observado na figura 27.

Figura 27 - Página de produtos



Fonte: Elaborado pelo autor

Quando o usuário preenche o campo de busca e clica no botão “Buscar”, o valor informado é processado por um código-fonte que não faz qualquer tipo de filtro e exibe o filtro informado no campo de busca na tela da página de resultados, conforme pode ser observado na figura 28.

Figura 28 - Código de processamento da busca

```

15         <?php
16             $sql = "select * from produtos where titulo "
17                 . "like '%" . $_GET['busca'] . "%'";
18             $result = $conn->query($sql);
19         ?>
20         <h1 class="title">
21             Resultados da Busca: <i>"<?php echo $_GET['busca']; ?>"</i>
22         </h1>

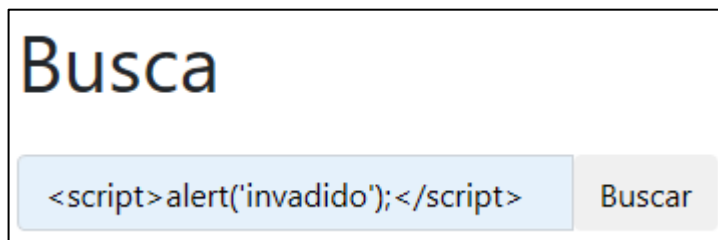
```

Fonte: Elaborado pelo autor

O código-fonte presente na figura 28 realiza uma busca no banco de dados passando a variável de busca para a consulta, em seguida exibe o valor na tela sem qualquer tipo de filtragem. Caso um usuário mal-intencionado queira explorar essa

vulnerabilidade, é possível informar um script malicioso no campo de busca como pode ser observado na figura 29.

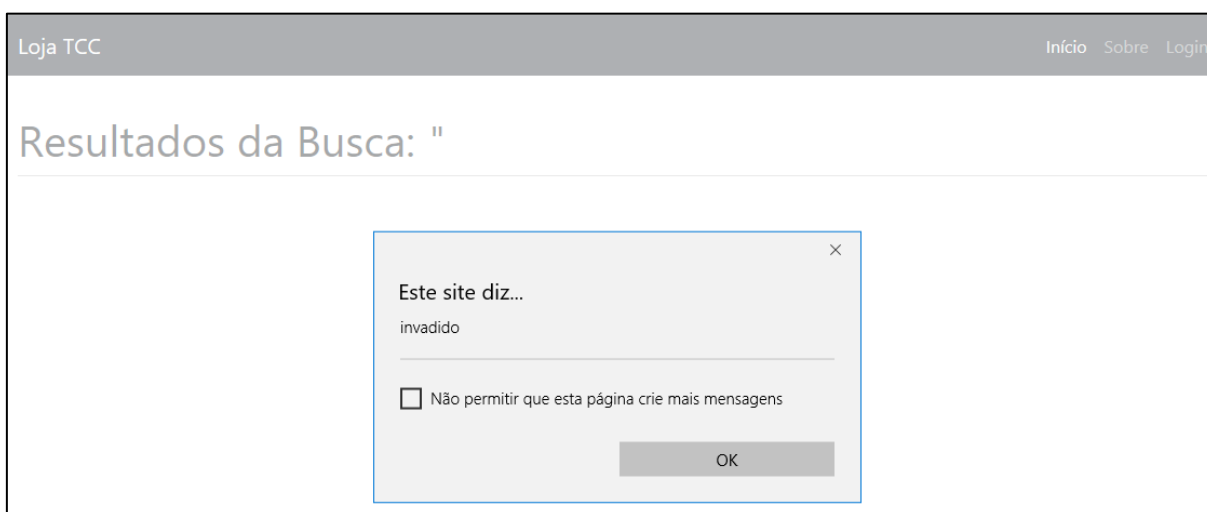
Figura 29 - Script malicioso campo busca



Fonte: Elaborado pelo autor

Ao clicar no botão “Buscar” o valor será passado para a URL “busca.php?busca=%3Cscript%3Ealert%28%27invadido%27%29%3B%3C%2Fscript%3E” e então será refletido na página de resultados da busca conforme pode ser observado na figura 30.

Figura 30 - Resultado da busca com script malicioso



Fonte: Elaborado pelo autor

Conforme apresentado na figura 30, é possível perceber que o script malicioso foi refletido na página web. Caso o atacante quisesse, poderia fornecer a URL infectada para um usuário comum, que, por sua vez, irá executá-la e será infectado pelo script malicioso.

Na próxima seção serão abordadas as formas de prevenção que poderiam ser adotadas para que este tipo de ataque não ocorresse.

5.2.2 Formas de prevenção

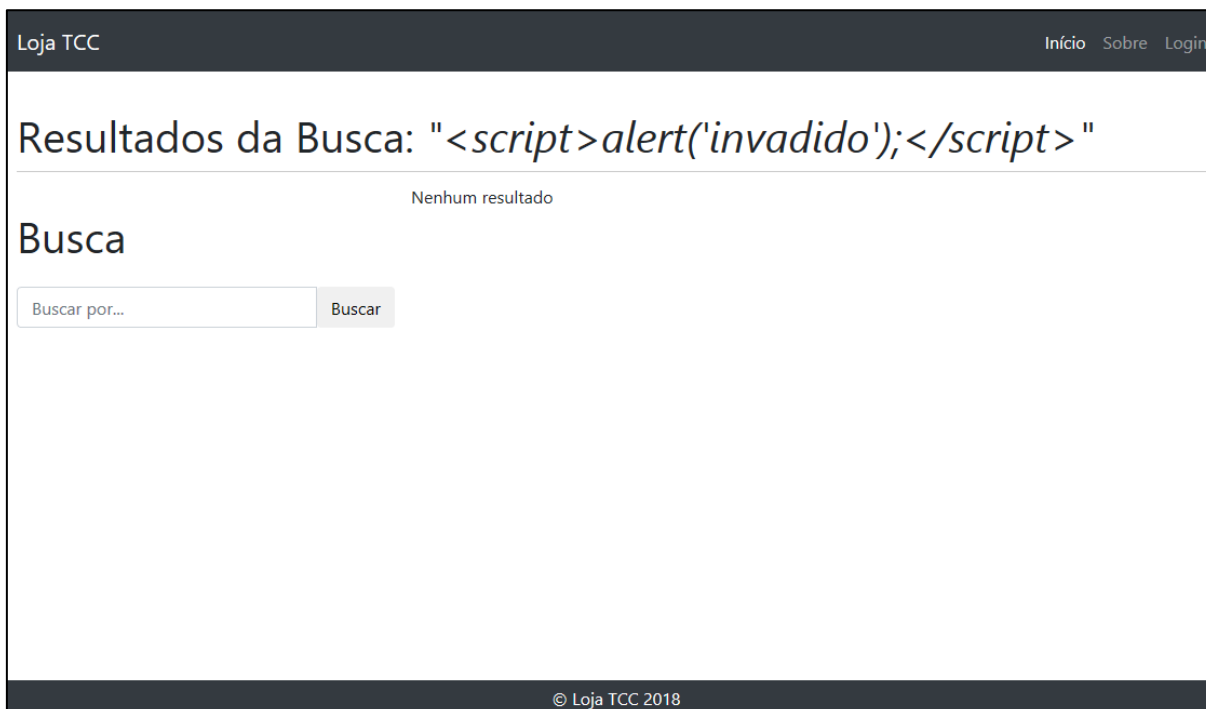
A fim de prevenir o ataque de XSS refletido na página de buscas seria possível aplicar um filtro na variável de busca impedindo a interpretação de determinados caracteres pelo sistema. A linguagem PHP possui uma função chamada *htmlentities* que realiza a conversão de qualquer caractere especial para a codificação entidade. Desta forma, ao invés da página executar o script irá apenas escrevê-lo como texto puro na página. A alteração do código-fonte contendo a correção indicada pode ser observada conforme figura 31.

Figura 31 - Código de processamento da busca com filtro

```
15     <?php
16         $sql = "select * from produtos where titulo "
17             . "like '%" . $_GET['busca'] . "%'";
18         $result = $conn->query($sql);
19     ?>
20     <h1 class="title">
21         Resultados da Busca: <i>"<?php echo htmlentities($_GET['busca']); ?>"</i>
22     </h1>
```

Fonte: Elaborado pelo autor

Como resultado da aplicação do filtro, ao informar o script malicioso ele será exibido na tela não causando mal algum, conforme pode ser observado na figura 32.

Figura 32 - Resultado da busca com filtro aplicado

Fonte: Elaborado pelo autor

5.3 Execução do ataque XSS armazenado

O XSS armazenado consiste na injeção de scripts maliciosos que são armazenados na aplicação comumente em bancos de dados do sistema. Nas próximas seções serão abordados os procedimentos de exploração do XSS armazenado na aplicação desenvolvida pelo autor, bem como as formas de prevenção da mesma.

5.3.1 Explorando a vulnerabilidade

Para exemplificar o XSS armazenado dentro da aplicação desenvolvida será utilizado a página de cadastro de usuários, a qual possui um formulário com os campos de nome, usuário e senha, conforme pode ser observado na figura 33.

Figura 33 - Página de cadastro de usuários

Loja TCC Início Sobre Login

Cadastro de Usuário

Nome

Usuário

Senha

© Loja TCC 2018

Fonte: Elaborado pelo autor

Quando o usuário preenche os campos do formulário de cadastro e clica no botão “Cadastre-se”, os valores informados são inseridos no banco de dados sem nenhum tipo de filtro. Após o sucesso no cadastro, o usuário é autenticado e redirecionado para a página minha conta, conforme o código-fonte que pode ser observado na figura 34.

Figura 34 - Código de inserção do usuário

```
2 session_start();
3 require_once 'config/mysql.conf.php';
4
5 $nome = $_POST['nome'];
6 $usuario = $_POST['usuario'];
7 $senha = $_POST['senha'];
8
9 if (empty($nome) || empty($usuario) || empty($senha)) {
10     header('location: usuario-cadastro-form.php?mensagem=1');
11 } else {
12     $sql = "insert into usuarios values(null, "
13         . " '" . $nome . "', "
14         . " '" . $usuario . "', "
15         . " '" . $senha . "') ";
16
17     $query = $conn->query($sql);
18     $_SESSION["logado"] = true;
19     $_SESSION["id_usuario"] = $conn->insert_id;
20     header('location: admin/minha-conta.php');
21 }
```

Fonte: Elaborado pelo autor

Por sua vez, o código-fonte presente na página minha conta recupera os dados do usuário cadastrado e os exibe na tela sem nenhum tipo de filtragem de dados, conforme pode ser observado na figura 35.

Figura 35 - Código de exibição de dados minha conta

```
20         <?php
21         $sql = "select * from usuarios where id = "
22             . $_SESSION["id_usuario"] . """;
23         $result = $conn->query($sql);
24         $row = $result->fetch_array()
25         ?>
26         <h1>Minha Conta</h1>
27         <div class="jumbotron">
28             <h1>Bem-Vindo</h1>
29             <p>
30                 <strong>Nome:</strong>&nbsp;
31                 <span><?php echo $row['nome']; ?></span>
32             </p>
33             <p>
34                 <strong>Usuário:</strong>&nbsp;
35                 <span><?php echo $row['usuario']; ?></span>
36             </p>
37         </div>
```

Fonte: Elaborado pelo autor

A figura 36 contém a página web que representa a tela minha conta.

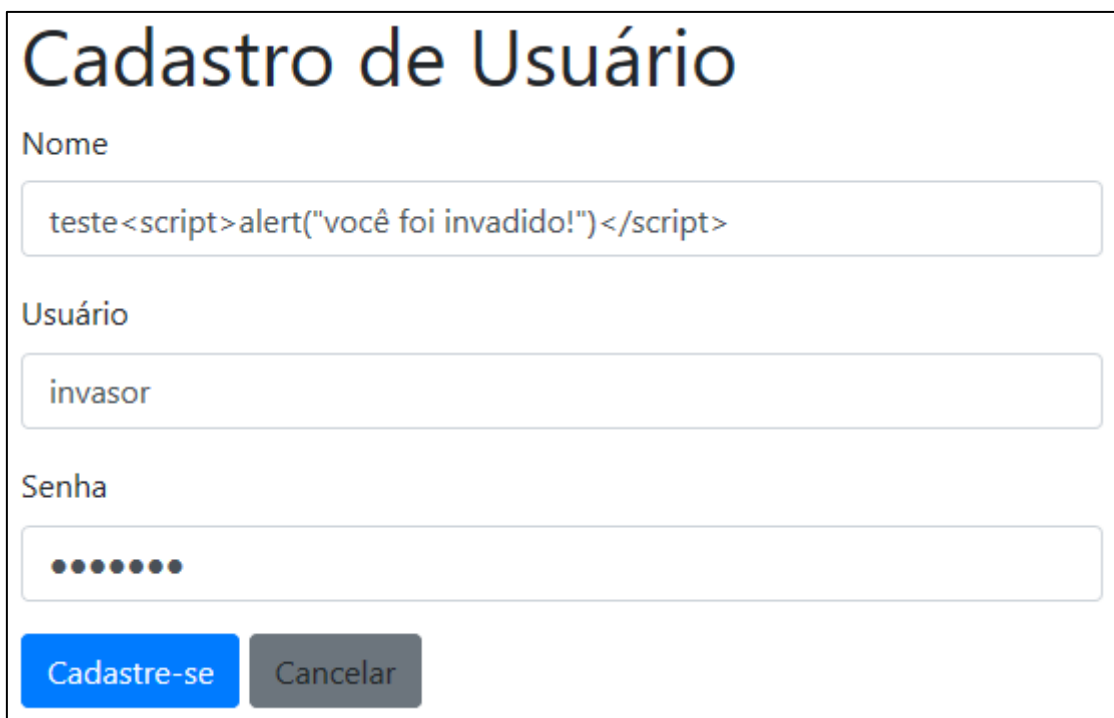
Figura 36 - Página minha conta



Fonte: Elaborado pelo autor

Caso um atacante queira explorar essa vulnerabilidade, pode informar um script malicioso nos campos do formulário de cadastro de usuário, como pode ser observado na figura 37.

Figura 37 - Script malicioso cadastro do usuário



Cadastro de Usuário

Nome
teste<script>alert("você foi invadido!")</script>

Usuário
invasor

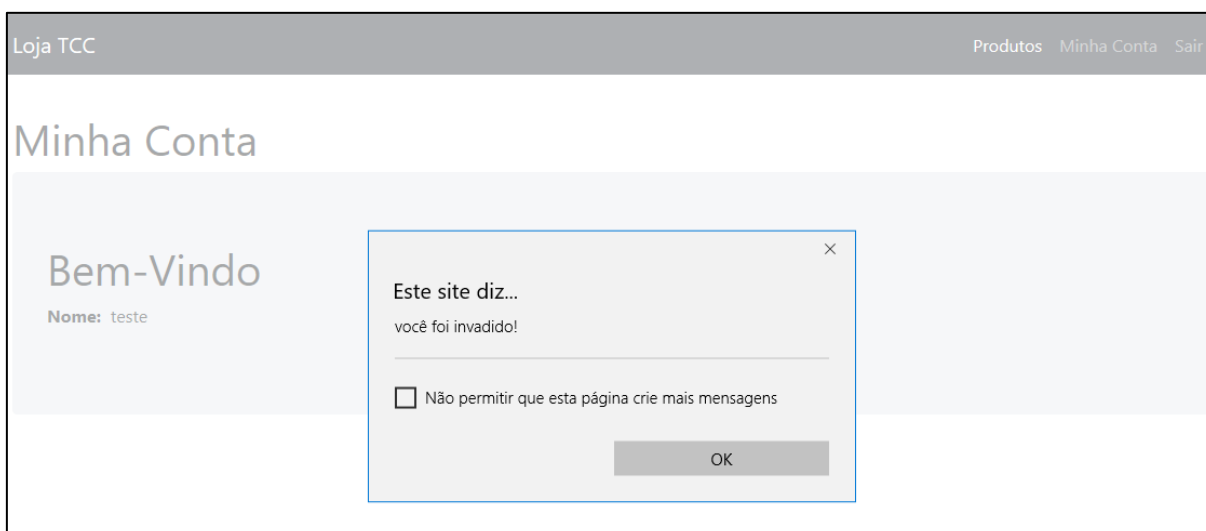
Senha
●●●●●●

Cadastre-se Cancelar

Fonte: Elaborado pelo autor

Na figura 38 percebe-se que após clicar no botão “Cadastre-se”, os dados serão armazenados no banco de dados, assim o código malicioso será sempre executado.

Figura 38 - Página minha conta com script malicioso



Loja TCC Produtos Minha Conta Sair

Minha Conta

Bem-Vindo
Nome: teste

Este site diz...
você foi invadido!

Não permitir que esta página crie mais mensagens

OK

Fonte: Elaborado pelo autor

Conforme apresentado na figura 38, o script malicioso é recuperado do banco de dados e sempre será executado na página web. O XSS armazenado é o mais perigoso, pois os scripts maliciosos ficam hospedados na aplicação invadida.

Na próxima seção serão abordadas as formas de prevenção que poderiam ser adotadas para que este tipo de ataque não ocorresse.

5.3.2 Formas de prevenção

A fim de prevenir o ataque de XSS armazenado nas páginas de cadastro de usuário e minha conta é possível aplicar um filtro nas variáveis de entrada que irão ser armazenadas no banco de dados no cadastro do usuário e nas variáveis que são exibidas para o usuário na página minha conta. Desta forma, pode-se impedir a interpretação de caracteres maliciosos nestas páginas.

Assim como no XSS refletido, podemos utilizar a função *htmlentities* da linguagem PHP para realizar a conversão de dos caracteres especiais contidos nos códigos maliciosos para a codificação entidade.

A alteração do código-fonte contendo a correção indicada para a página de cadastro do usuário pode ser observada conforme figura 39.

Figura 39 - Código de inserção do usuário com filtro

```

2  session_start();
3  require_once 'config/mysql.conf.php';
4
5  $nome = $_POST['nome'];
6  $usuario = $_POST['usuario'];
7  $senha = $_POST['senha'];
8
9  if (empty($nome) || empty($usuario) || empty($senha)) {
10     header('location: usuario-cadastro-form.php?mensagem=1');
11 } else {
12     $sql = "insert into usuarios values(null, "
13         . " '" . htmlentities($nome) . "', "
14         . " '" . htmlentities($usuario) . "', "
15         . " '" . htmlentities($senha) . "') ";
16
17     $query = $conn->query($sql);
18     $_SESSION["logado"] = true;
19     $_SESSION["id_usuario"] = $conn->insert_id;
20     header('location: admin/minha-conta.php');
21 }
22 ?>

```

Fonte: Elaborado pelo autor

O código-fonte apresentado na figura 40 contém a correção indicada para a página minha conta.

Figura 40 - Código de exibição de dados minha conta com filtro

```

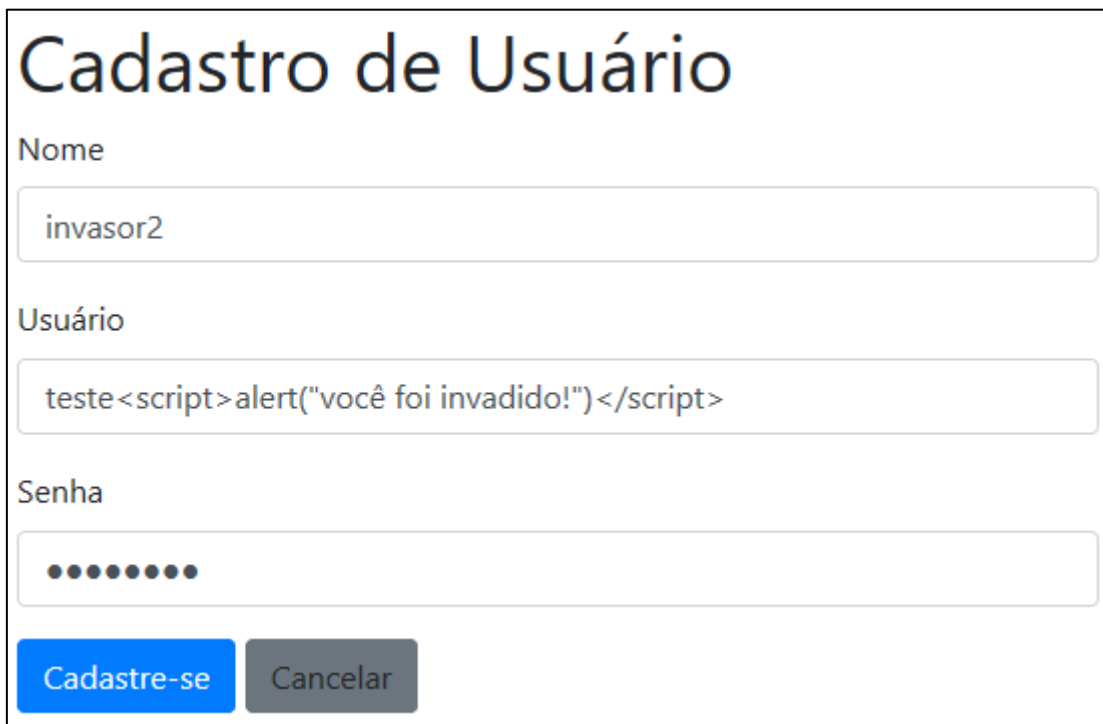
29     <p>
30         <strong>Nome:</strong>&nbsp;
31         <span><?php echo htmlentities($row['nome']); ?></span>
32     </p>
33     <p>
34         <strong>Usuário:</strong>&nbsp;
35         <span><?php echo htmlentities($row['usuario']); ?></span>
36     </p>

```

Fonte: Elaborado pelo autor

Na figura 41 o script malicioso é novamente informado no cadastro de usuários.

Figura 41 - Script malicioso no cadastro do usuário após aplicação dos filtros



Cadastro de Usuário

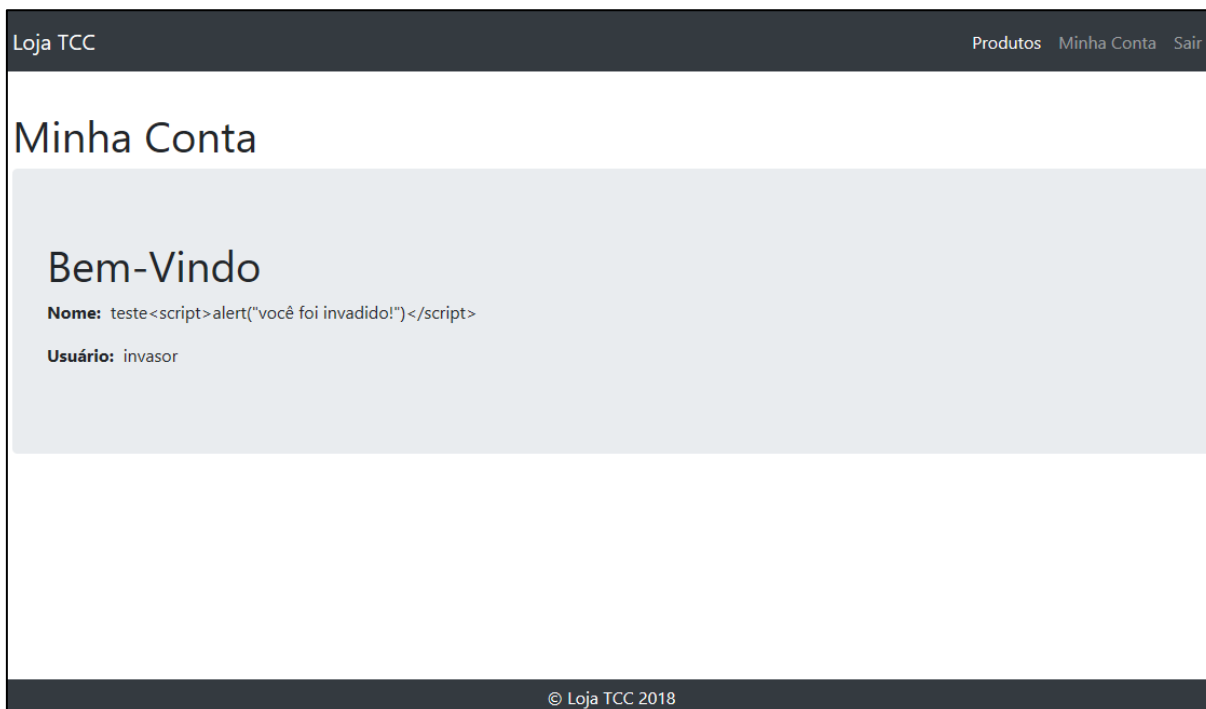
Nome

Usuário

Senha

Fonte: Elaborado pelo autor

Porém o script malicioso não surtirá efeitos, devido a aplicação dos filtros adequados presentes nos códigos das figuras 39 e 40, como pode ser observado na figura 42.

Figura 42 - Página minha conta com filtro aplicado

Fonte: Elaborado pelo autor

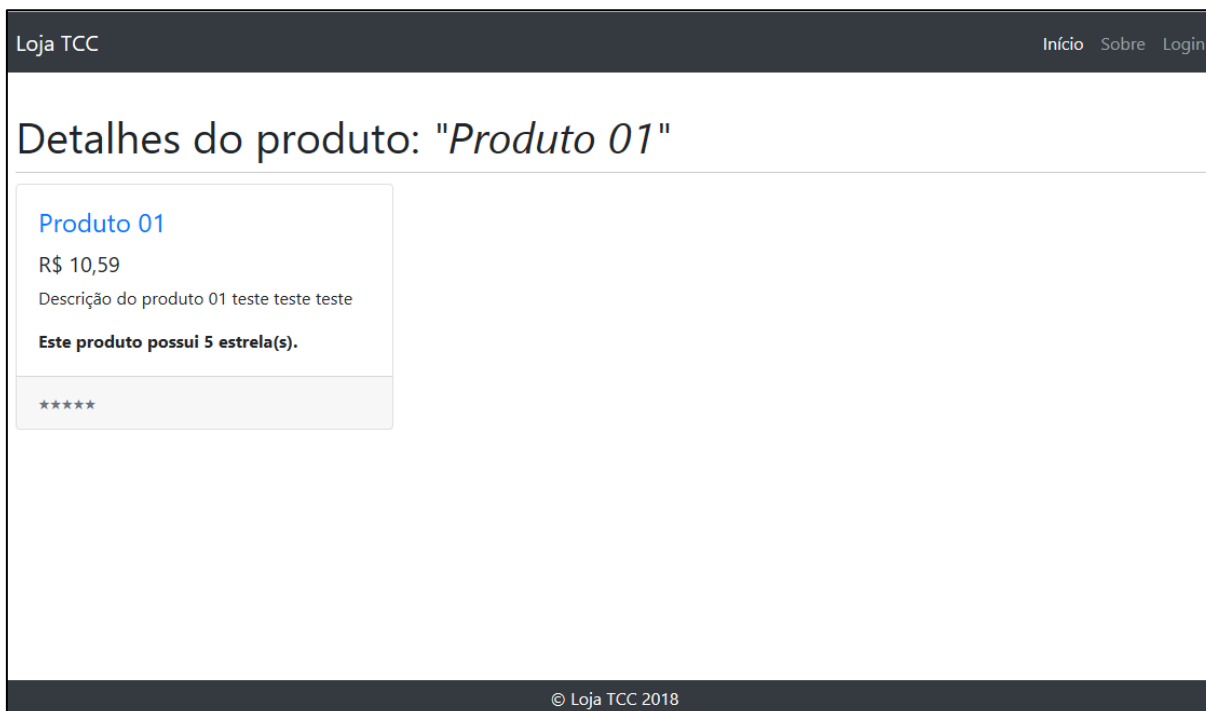
5.4 Execução do ataque XSS baseado em DOM

O XSS baseado em DOM é bastante similar ao XSS refletido, porém a maior diferença deste é que o script malicioso não é repassado para o lado do servidor da aplicação. Ele consiste na injeção de scripts maliciosos que são refletidos na aplicação através da árvore do documento DOM. Nas próximas seções serão abordados os procedimentos de exploração deste tipo de XSS na aplicação desenvolvida pelo autor, bem como as formas de prevenção da mesma.

5.4.1 Explorando a vulnerabilidade

Para exemplificar o XSS baseado em DOM dentro da aplicação desenvolvida será utilizado a página de detalhes do produto a qual contém os dados de um determinado produto selecionado previamente, conforme pode ser observado na figura 43.

Figura 43 - Página de detalhes do produto



Fonte: Elaborado pelo autor

A página apresentada na figura 43 contém um componente para apresentar a quantidade de estrelas do produto em questão. Este componente é implementado utilizando a linguagem de programação Javascript. Ao acessar a URL “produto.php?estrelas=5&idProduto=1”, o script recupera o parâmetro “estrelas” através do método GET e apresenta o texto “Este produto possui 5 estrela(s)” na página de detalhes do produto, conforme pode ser observado na figura 44.

Figura 44 - Código de geração da quantidade de estrelas

```
50     <script>
51     var url = document.URL;
52     var pos = url.indexOf("estrelas=");
53     if (pos != -1) {
54         var numeroDeEstrelas = url.substring(pos + 9, url.length).split('&')[0];
55         var texto = "Este produto possui " + numeroDeEstrelas + " estrela(s).";
56         document.getElementById("estrelas").innerHTML = unescape(texto);
57     }
58     </script>
```

Fonte: Elaborado pelo autor

O código-fonte presente na figura 44 obtém os valores da URL sem qualquer tipo de filtro. Deste modo, caso um usuário mal-intencionado queira explorar esta

vulnerabilidade pode fazê-la informando um script malicioso na URL da aplicação, conforme pode ser observado na figura 45.

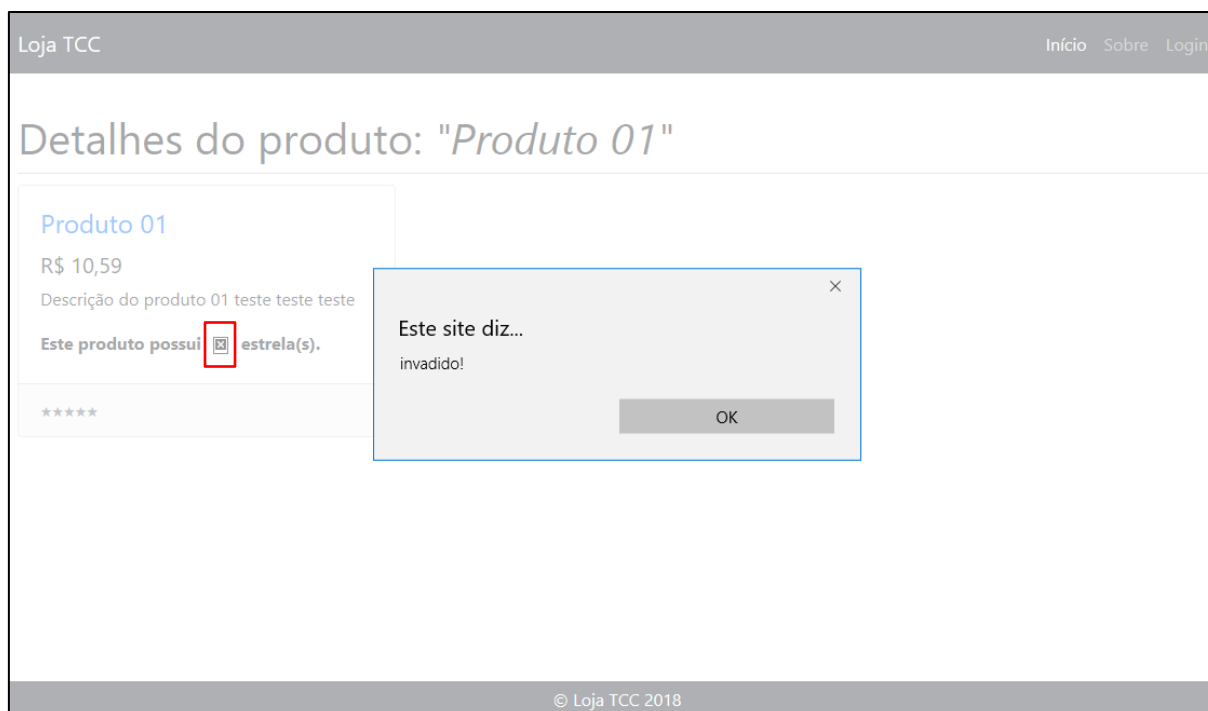
Figura 45 - Script malicioso na URL da página de detalhes do produto

```
33
34
35     ?estrelas=&idProduto=1
36
37
```

Fonte: Elaborado pelo autor

Ao informar a *tag* `` com uma imagem inexistente, será disparado o evento *onerror* da mesma *tag*. Sendo assim será disparado também o script malicioso na página, conforme pode ser observado na figura 46.

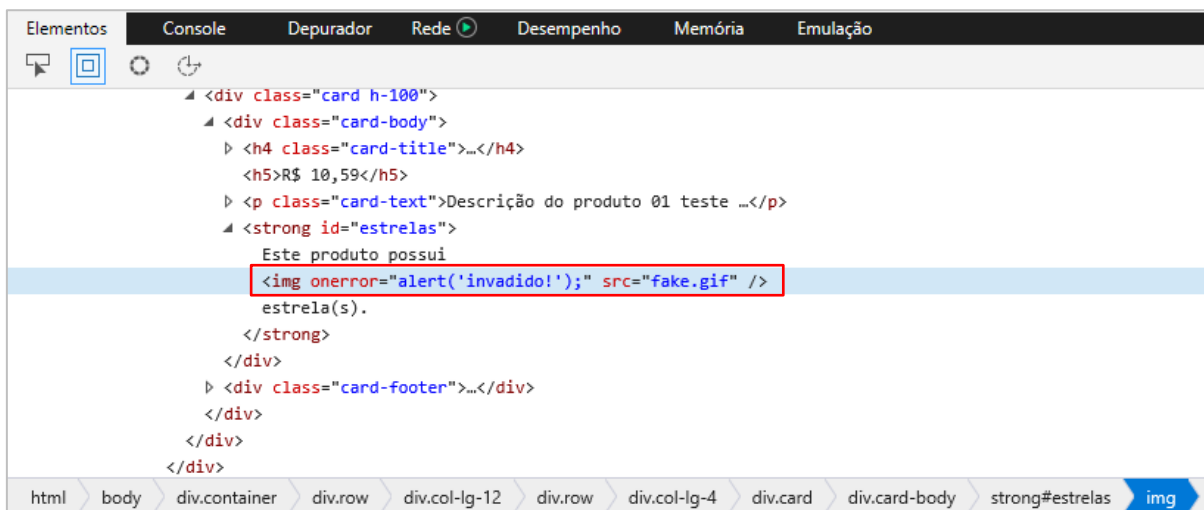
Figura 46 - Página de detalhes do produto com script malicioso



Fonte: Elaborado pelo autor

Conforme apresentado na figura 46, o script malicioso foi executado na página web. A figura 47 mostra o código que foi injetado no DOM.

Figura 47 - Código injetado por meio do XSS DOM



```
Elementos Console Depurador Rede Desempenho Memória Emulação
<div class="card h-100">
  <div class="card-body">
    <h4 class="card-title">...</h4>
    <h5>R$ 10,59</h5>
    <p class="card-text">Descrição do produto 01 teste ...</p>
    <strong id="estrelas">
      Este produto possui
      
      estrela(s).
    </strong>
  </div>
  <div class="card-footer">...</div>
</div>
```

Fonte: Elaborado pelo autor

Utilizando esta mesma técnica o atacante poderia enviar a URL infectada para outro usuário com o intuito de obter seu cookie de sessão ou qualquer outro dado do usuário através do script malicioso.

Na próxima seção serão abordadas as formas de prevenção que poderiam ser adotadas para que este tipo de ataque não ocorresse.

5.4.2 Formas de prevenção

A fim de prevenir o ataque de XSS baseado em DOM na página de detalhes do produto seria possível aplicar um filtro na variável que recupera os valores da URL presente no código-fonte da página. Assim, os dados maliciosos ficariam codificados e não seriam mais executados. Para realizar tal alteração basta utilizar uma função da linguagem Javascript chamada *encodeURIComponent*, ela utiliza a codificação percentual para converter os caracteres especiais. A alteração do código-fonte contendo a correção indicada pode ser observada conforme figura 48.

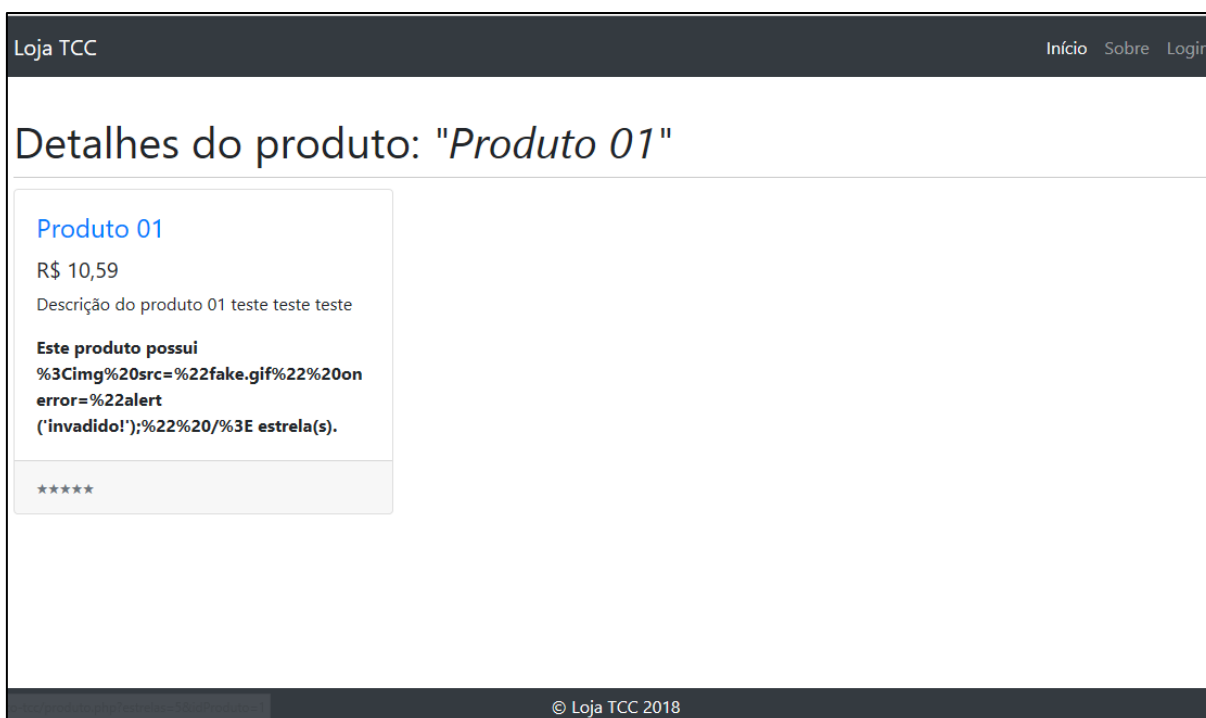
Figura 48 - Código de geração da quantidade de estrelas com filtro

```
50     <script>
51     var url = encodeURI(document.URL);
52     var pos = url.indexOf("estrelas=");
53     if (pos != -1) {
54         var numeroDeEstrelas = url.substring(pos + 9, url.length).split('&')[0];
55         var texto = "Este produto possui " + numeroDeEstrelas + " estrela(s).";
56         document.getElementById("estrelas").innerHTML = unescape(texto);
57     }
58     </script>
```

Fonte: Elaborado pelo autor

Como resultado da aplicação do filtro na URL do script em questão, ao informar novamente o script malicioso na URL, ele será exibido na tela não causando mal algum, conforme pode ser observado na figura 49.

Figura 49 - Página de detalhes do produto com filtro aplicado



The screenshot shows a web page titled "Loja TCC" with navigation links for "Início", "Sobre", and "Login". The main heading is "Detalhes do produto: 'Produto 01'". The product details are displayed in a box:

- Produto 01**
- R\$ 10,59
- Descrição do produto 01 teste teste teste
- Este produto possui**
- `%3Cimg%20src=%22fake.gif%22%20onerror=%22alert('invadido!');%22%20/%3E` estrela(s).
- *****

At the bottom of the page, there is a footer: "© Loja TCC 2018".

Fonte: Elaborado pelo autor

6 PREVENÇÕES AO ATAQUE XSS

Neste capítulo serão abordados alguns conceitos importantes no contexto da prevenção dos ataques de XSS.

6.1 Prudência na inserção de dados sensíveis

Em linhas gerais, um dos grandes problemas nos incidentes de segurança da informação é a falta de prudência dos usuários da aplicação. Muitos dos eventos de falha de segurança e vazamento de informações confidenciais ocorrem devido à falta de prudência do usuário na inserção de seus dados.

Algumas dicas são muito relevantes para a proteção dos dados do usuário na internet, em especial com relação a prevenção do XSS:

- Analisar cada URL antes mesmo de ser clicada.
- Jamais passar dados sensíveis como dados de cartão de crédito e senhas através de contato por e-mail ou telefone.
- Caso suspeite que algum e-mail se trata de um golpe, o usuário nunca deve responder tal e-mail e jamais clicar em links contidos no e-mail.
- Evitar acessar sites que detém informações sigilosas através de computadores de terceiros.

A conscientização dos usuários é a maneira mais eficiente de se mitigar os riscos de segurança em aplicações web. Um usuário consciente que sabe como se comportar diante de uma fraude jamais cairá em golpes da internet.

6.2 Utilização de codificações no desenvolvimento de aplicações

O desenvolvimento de aplicações de Internet é frequentemente o local onde são introduzidas as principais vulnerabilidades. Muitos incidentes de segurança de captura de senha eletrônica foram devidos diretamente à segurança inadequada da aplicação. (SCHETINA; GREEN; CARLSON, 2002, p.385).

O papel do desenvolvedor é de extrema importância no desenvolvimento de aplicações seguras. A fim de evitar os ataques de XSS o desenvolvedor deve sempre realizar as devidas codificações para todas as variáveis de entrada e saída da aplicação conforme pôde ser visto nas seções do capítulo 5 deste trabalho.

A Owasp define algumas regras de prevenção contra os ataques de XSS (*Cross Site Scripting*) a fim de orientar o desenvolvimento de aplicações web.

De acordo com Owasp (2018) estas regras:

“[...] destinam-se a impedir todo o XSS em seu aplicativo. Embora essas regras não permitam a liberdade absoluta de colocar dados não confiáveis em um documento HTML, elas devem cobrir a grande maioria dos casos de uso comuns. Você não precisa permitir todas as regras da sua organização. Muitas organizações podem achar que permitir somente a Regra nº 1 e Regra nº 2 é suficiente para suas necessidades.[..].”

Grande parte destas regras consiste em “escapar” ou seja, codificar os elementos HTML a fim de prevenir os ataques, como pode ser observado a seguir:

- **Regra 0** - Nunca insira dados não confiáveis exceto em locais permitidos:

Esta regra diz que sempre que for possível, o desenvolvedor não deve permitir a entrada de dados não confiáveis em uma aplicação ou em seu código HTML. (OWASP, 2018)

- **Regra 1** - Escapar HTML antes de inserir dados não confiáveis no conteúdo do elemento HTML:

Esta regra é importante para quando se deseja colocar dados não confiáveis diretamente no corpo HTML em algum lugar. A maioria dos frameworks web tem um método para HTML escapar para os caracteres detalhados, porém não é suficiente e o desenvolvedor deve realizar o seu trabalho. (OWASP, 2018)

- **Regra 2** - Escapar atributos antes de inserir dados não confiáveis em atributos comuns de HTML:

Esta regra diz que é importante manter todos os dados inseridos na aplicação codificados. Valores de atributos e elementos injetados em uma aplicação devem ter seus caracteres especiais codificados. (OWASP, 2018)

- **Regra 3** - Escapar javascript antes de inserir dados não confiáveis em valores de dados no javascript.

Esta regra diz respeito aos códigos escritos na linguagem de programação Javascript e aos seus manipuladores de eventos.

Ela diz que o único local seguro para colocar dados não confiáveis nesse código está dentro de um "valor de dados" entre aspas. Incluir dados não confiáveis dentro de qualquer outro contexto Javascript é bastante perigoso (OWASP, 2018)

Nesta seção foram abordadas práticas importantes no desenvolvimento seguro a fim de mitigar os riscos de um ataque de XSS na aplicação. O desenvolvedor deve fazer bom uso das linguagens de programação que em sua grande maioria possuem funções para tratamento de dados e *escape* de código malicioso, também é importante o uso de bibliotecas de codificação segura. Na próxima seção este assunto será abordado com maior profundidade.

6.3 Utilização de bibliotecas para codificação segura

A maioria das linguagens de programação possuem recursos nativos da linguagem para tratamento de dados e sanitização de variáveis para prevenção de ataques, no entanto em algumas ocasiões específicas podem haver dificuldades ao validar a entrada de dados na aplicação. Desta forma é interessante a utilização de bibliotecas de codificação segura para facilitar a limpeza das entradas do usuário.

O uso de bibliotecas de codificação é uma recomendação da Owasp (2018). Existem inclusive a recomendação de algumas delas como o *HtmlSanitizer*, a biblioteca *Anti-XSS Library* para o desenvolvimento de aplicações na plataforma Microsoft, o *Java HTML Sanitizer Project* para o desenvolvimento de aplicações na plataforma Java, dentre outras.

Apesar das recomendações fornecidas pela Owasp e das qualidades apresentadas nesta seção, o uso de bibliotecas complementa a utilização dos recursos já existentes na linguagem de programação. Desta forma é importante que o desenvolvedor avalie a efetividade da utilização de tais bibliotecas para o contexto da aplicação que está sendo desenvolvida.

7 CONSIDERAÇÕES FINAIS

A utilização das aplicações web têm aumentado ao longo do tempo, assim como um grande volume de dados sigilosos tem sido trafegado na rede mundial de computadores, de forma que a preocupação com a segurança do usuário e sua privacidade é constante.

Neste contexto, manter as aplicações web seguras não é uma tarefa fácil, uma vez que, para cada tipo de vulnerabilidade de segurança é necessária uma ação específica correspondente para evitar a indisponibilidade das aplicações ou até mesmo vazamento de informações sigilosas do usuário.

O presente trabalho teve como objetivo principal a análise de vulnerabilidade do XSS (*Cross Site Scripting*), quais são as formas de ataque e prevenção. Para tal, abordou cada tipo de XSS separadamente e utilizou-se de uma aplicação web desenvolvida pelo autor a fim de verificar as vulnerabilidades de XSS existentes, bem como propor técnicas para solucioná-las.

Os resultados obtidos após o estudo da aplicação e das técnicas de codificação segura possibilitam ao leitor deste trabalho adquirir um amplo conhecimento sobre a importância deste tipo de ataque no contexto de segurança de aplicações, assim como entender de forma aprofundada a vulnerabilidade e as formas de prevenção da mesma.

Ainda no contexto de segurança em aplicações web, o trabalho possibilita a extensão do assunto abordando outras vulnerabilidades presentes nas aplicações. A Owasp define diversas vulnerabilidades como *Injection*, *SQL Injection*, *Sensitive Data Exposure*, dentre outras. Todas elas podem ser objeto de estudo para trabalhos posteriores a este.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 27002**: Tecnologia da informação, técnicas de segurança, código de prática para a gestão da segurança da informação. 1 ed. Rio de Janeiro, 2005. 120 p.

CONHECIMENTO GERAL. **Tabela ascii**, 2016. Disponível em: <https://www.conhecimentogeral.inf.br/tabela_ascii/> Acesso em: 14 maio 2018.

CERT.BR. **Estatísticas dos Incidentes reportados ao CERT.BR**, 2017. Disponível em: <<https://www.cert.br/stats/incidentes/>> Acesso em: 10 maio 2018.

FARIA, S. V. **Tabela ASCII completa**, 2011. Disponível em: <<https://desenvolvedorinteroperavel.wordpress.com/2011/09/11/tabela-ascii-completa/>> Acesso em: 11 jun 2018.

G1 GLOBO. **Falhas mostram despreparo de sites de redes sociais**, 2010. Disponível em: <<http://g1.globo.com/tecnologia/noticia/2010/09/falhas-mostram-despreparo-de-sites-de-redes-sociais.html> /> Acesso em: 10 maio 2018.

GROSSMAN, J. *et al.* **XSS attacks**: cross site scripting exploits and defense. Burlington: Elsevier, Inc, 2007.

KALLIN, Jakob, VALBUENA, L. I. **Excess XSS**: comprehensive tutorial on cross-site scripting, 2013. Disponível em: <<https://excess-xss.com/>> Acesso em: 06 maio 2018.

LINHA DEFENSIVA. **Falha no site do bradesco permitiu ataque XSS**, 2008. Disponível em: <<https://linhadefensiva.org/2008/07/29/bradesco-pesquisa-inst-xss/>> Acesso em: 10 maio 2018.

MOZILLA. **O que é javascript**, 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/O_que_e_JavaScript> Acesso em: 06 maio 2018.

MOZILLA. **Modelo de objeto de documento (DOM)**, 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM/> Acesso em: 09 maio 2018.

OWASP. **Cross-site scripting (XSS)**, 2018. Disponível em: <[https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))> Acesso em: 07 maio 2018.

OWASP. **Types of cross-site scripting**, 2017. Disponível em: <https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting/> Acesso em: 10 maio 2018.

OWASP. **XSS (Cross Site Scripting) prevention cheat sheet**, 2018. Disponível em: <[https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet/](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet/)> Acesso em: 13 jun 2018.

PTSECURITY. **Web application attack statistics: Q1 2017**, 2017. Disponível em: <<https://www.ptsecurity.com/upload/corporate/ww-en/analytics/WebApp-Attacks-2017-eng.pdf/>> Acesso em: 09 jun 2018.

REDE SEGURA. **Série ataques: Saiba mais sobre o cross-site scripting (XSS)**, 2012. Disponível em: <<http://www.redesegura.com.br/2012/01/saiba-mais-sobre-o-cross-site-scripting-xss/>> Acesso em: 08 maio 2018.

SCHETINA, E.; GREEN, K.; CARLSON, J. **Aprenda a desenvolver e construir sites seguros**. Rio de Janeiro: Editora Campus Ltda, 2002.

SÊMOLA, Marcos. **Gestão da segurança da informação: uma visão executiva**. 2. ed. Rio de Janeiro: Elsevier Editora, 2014.

TODESCHINI, Leonardo, COSTA, G. R. **Como programar usando ferramentas livres: HTML, JavaScript, Apache, MySQL e PHP**. Rio de Janeiro: Editora Alta Books Ltda, 2006.

UTO, Nelson. **Teste de invasão de aplicações web**. Rio de Janeiro: Escola Superior de Redes, 2013.

WINCKLER, Marco, PIMENTA, M. S. **Avaliação de usabilidade de sites web**, 2002. Disponível em: <<https://www.irit.fr/~Marco.Winckler/2002-winckler-pimenta-ERI-2002-cap3.pdf/>> Acesso em: 04 maio 2018.