Uma ferramenta adaptativa para detecção de ataques *Cross-Site Scripting* no lado do cliente

Isabela M. M. Alves[1],
Julia A. Fernandez e Souza[1,2], **Dalbert M. Mascarenhas**[1], **Igor M. Moraes**[2]

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - CEFET/RJ
Petrópolis - RJ - Brasil

[2]Laboratório MidiaCom – IC/TCC/PGC
Universidade Federal Fluminense (UFF), Niterói – RJ – Brasil

isabela.alves@aluno.cefet-rj.br, juliabbud@midiacom.uff.br
dalbert.mascarenhas@cefet-rj.br, igor@ic.uff.br,

Abstract. *This paper proposes a tool for detecting Cross-site Scripting (XSS) attacks that operates directly on the client side and employs artificial intelligence techniques. The tool is implemented as a web browser extension and combines feature engineering, text vectorization, and the Random Forest algorithm, resulting in a robust model capable of identifying malicious patterns with low response time and high accuracy. The model used by the tool is trained and evaluated on a dataset constructed in this work from real web traffic and public payloads comprising different types of XSS attacks. By operating on the client side, the tool enables model retraining and adaptation to the user's browsing profile through HTTP traffic capture, which contributes to a significant reduction in false positives. The results achieved by the proposed tool exceed 99% in the main performance metrics, demonstrating its feasibility for protecting users against XSS attacks.*

Resumo. *Este artigo propõe uma ferramenta para detecção de ataques do tipo Cross-site Scripting (XSS) que é executada diretamente no lado do cliente e usa técnicas de inteligência artificial. A ferramenta é implementada como uma extensão de um navegador Web e combina engenharia de atributos, vetorização textual e o algoritmo Random Forest, resultando em um modelo robusto capaz de identificar padrões maliciosos com baixo tempo de resposta e alta acurácia. O modelo usado pela ferramenta é treinado e avaliado com um conjunto de dados construído neste artigo a partir de coletas de tráfego real e cargas úteis (payloads) públicas compostas de diferentes tipos de ataques XSS. Por operar no lado do cliente, a ferramenta permite o retreinamento e a adaptação do modelo ao perfil de navegação do usuário por meio da captura de tráfego HTTP, o que contribui para a redução significativa de falsos positivos. Os resultados obtidos com a ferramenta superam 99% nas principais métricas de desempenho comprovando a viabilidade da ferramenta proposta para a proteção de usuários contra ataques XSS.*

1. Introdução

O *Cross-site Scripting* (XSS) está entre as vulnerabilidades mais recorrentes em aplicações Web [Liu et al., 2019]. Ao longo das edições do OWASP Top 10 de 2004,

2007, 2010, 2013 e 2017, o XSS figurou, respectivamente, nas posições 4ª, 4ª, 1ª, 3ª e 7ª, evidenciando sua relevância histórica e persistência [Liu et al., 2019]. Nas edições mais recentes, a vulnerabilidade continuou em destaque ao ser classificada como ataque de injeção, ocupando a 3ª posição em 2021 [OWASP Foundation, 2021a] e a 5ª posição em 2025 [OWASP Foundation, 2021b].

O XSS permite a injeção de código malicioso, geralmente *scripts* em *JavaScript*, que será executado no navegador do usuário. A exploração ocorre por meio de entradas não validadas ou não codificadas corretamente, como parâmetros em URLs (GET), campos de formulários (POST), cabeçalhos HTTP, *cookies* e *uploads* de arquivos [Bastos et al., 2025]. O código malicioso injetado é interpretado pelo navegador como legítimo, tornando a própria aplicação Web uma cúmplice involuntária do ataque [Bastos et al., 2025]. Este ataque pode ser classificado em três tipos principais: refletido (ou não-persistente), armazenado (ou persistente) e baseado em DOM. Cada um possui características específicas quanto à forma de injeção e persistência do código malicioso [Bastos et al., 2025].

Métodos tradicionais de mitigação, como filtros baseados em regras, validação de entrada e *Content Security Policy* (CSP), embora necessários, apresentam limitações frente a variantes ofuscadas, ataques *zero-day* e, especialmente, ao XSS baseado em *Document Object Model* (DOM), cuja execução ocorre exclusivamente no lado do cliente [Liu et al., 2019, Sarmah et al., 2018]. Nesse contexto, técnicas de aprendizado de máquina têm se mostrado promissoras para identificação automática de ataques XSS, explorando abordagens baseadas em análise de padrões, engenharia de atributos e vetorização textual [Kumar e Ponsam, 2023, Prasetio et al., 2021, Thajeel et al., 2023]. Apesar dos avanços, grande parte dessas soluções concentra-se em cenários *server-side* ou em ambientes controlados, não contemplando adequadamente a execução dos ataques no lado do cliente.

Diante desse cenário, este artigo propõe a ferramenta *User-side Adaptive XSS Detector* (UAXD) voltada ao usuário final para detecção de ataques XSS, operando diretamente no lado do cliente por meio de uma extensão de navegador integrada a um classificador baseado em aprendizado de máquina. A abordagem prioriza a detecção de ataques XSS baseados em DOM — característicos do lado cliente — sem deixar de contemplar ataques refletidos e armazenados. Além disso, a ferramenta permite a captura de tráfego e o retreinamento do modelo de acordo com o perfil de navegação do usuário, contribuindo para a redução de falsos positivos.

A ferramenta é elaborada a partir de uma base de dados construída com tráfego real de navegação e *payloads* públicos de XSS, considerando diferentes cenários e tipos de ataque. Os resultados experimentais demonstram que o UAXD alcança desempenho superior a 99% nas principais métricas de avaliação, como acurácia, precisão, *recall* e *F1-score*, evidenciando sua eficácia na detecção de ataques XSS no lado do cliente.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a ferramenta proposta, incluindo a arquitetura da ferramenta, a construção da base de dados, os atributos e modelo utilizados e os componentes da UAXD. A Seção 4 discute os resultados experimentais. Por fim, a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Sarmah *et al.* realizam uma ampla revisão dos métodos de detecção de ataques XSS, categorizando estes ataques em três abordagens principais: análise estática, análise dinâmica e mecanismos híbridos [Sarmah et al., 2018]. O trabalho aponta que métodos estáticos permitem detecção precoce, mas apresenta limitações frente a ataques que empregam o código dinâmico e técnicas de ofuscação, enquanto métodos dinâmicos possuem maior capacidade de detecção em tempo de execução, porém com custo computacional elevado. As soluções híbridas oferecem melhor equilíbrio entre precisão e desempenho, embora ainda enfrentem desafios de escalabilidade e adaptação a aplicações modernas baseadas em *JavaScript* e manipulação do DOM.

Liu *et al.* [Liu et al., 2019] aprofundam a análise apresentada por Sarmah *et al.* ao propor uma taxonomia detalhada das vulnerabilidades XSS, classificando-as em ataques refletidos, armazenados e baseados em DOM. Os autores demonstram que técnicas de análise estática, embora eficazes para falhas do lado do servidor, são ineficazes na detecção de XSS baseado em DOM, uma vez que esse tipo de ataque ocorre exclusivamente no ambiente do cliente. Além disso, o estudo aponta que métodos existentes ainda apresentam elevadas taxas de falsos positivos e dificuldades na identificação de *payloads* ofuscados.

No contexto de aprendizado de máquina, Kumar e Ponsam [Kumar e Ponsam, 2023] demonstram a eficácia do algoritmo *Random Forest* na detecção de diferentes ataques XSS, alcançando altos índices de acurácia, precisão e revocação (*recall*). Os autores ressaltam a importância do pré-processamento adequado dos dados, incluindo técnicas como tokenização, codificação e vetorização, para que o modelo consiga diferenciar *scripts* maliciosos de *scripts* benignos com maior eficácia. Contudo, o estudo reconhece que a construção de bases de dados robustas e representativas ainda é um desafio, especialmente frente à evolução constante dos vetores de ataque.

Prasetio *et al.* [Prasetio et al., 2021] exploram uma abordagem híbrida baseada em n-gramas e seleção de características sintéticas e meta-informações, obtendo acurácia de 99,87% e baixa taxa de falsos positivos, 0,039%. O estudo destaca que os ataques XSS possuem padrões específicos na sua estrutura de caracteres, os quais podem ser modelados por linguística computacional (n-gramas) e por atributos derivados, como número de eventos *JavaScript*, caracteres especiais, comprimento da *string*, entre outros. A pesquisa também reforça que abordagens puramente baseadas em seleção de características apresentam desempenho inferior (77,39%), evidenciando que a combinação de métodos é mais eficaz para capturar os padrões complexos presentes em *payloads* maliciosas.

Thajeel *et al.* [Thajeel et al., 2023] fizeram uma revisão sistemática de 52 trabalhos publicados entre 2018 e 2023, identificando crescimento expressivo no uso de técnicas de aprendizado de máquina e profundo para detecção de XSS. O estudo destaca a eficácia de modelos como *Random Forest*, CNN, LSTM e BiLSTM, bem como a relevância de estratégias robustas de pré-processamento. Apesar dos avanços, permanecem desafios relacionados à evasão adversarial e à necessidade de atualização contínua dos modelos.

Em síntese, a literatura indica que abordagens baseadas em inteligência artificial

são promissoras para a detecção de ataques XSS, especialmente quando combinadas a mecanismos dinâmicos de análise. No entanto, a maioria das soluções concentra-se no lado servidor (*server-side*) ou em ambientes controlados.

Estudos recentes passam a explorar a detecção diretamente no cliente, com foco na redução de falsos positivos e adaptação a cenários reais de navegação [Oshoiribhor e John-Otumu, 2025]. Ainda assim, tais abordagens apresentam limitações quanto à adaptação contínua ao perfil do usuário.

Nesse contexto, a ferramenta proposta diferencia-se por operar diretamente no navegador e incorporar um mecanismo de retreinamento, permitindo adaptação dinâmica ao comportamento de navegação, além de priorizar a detecção de ataques XSS baseados em DOM.

3. A Ferramenta Proposta

Esta seção apresenta a UAXD, uma ferramenta para detecção de ataques XSS no lado do cliente por meio de uma extensão de navegador integrada a um modelo de aprendizado de máquina, descrevendo sua arquitetura, mecanismos de coleta e processamento de dados, modelo de classificação e principais componentes.

3.1. Arquitetura da UAXD

A Figura 1 apresenta a arquitetura geral da ferramenta, destacando os principais módulos envolvidos no processo de extração de dados, classificação, filtragem de informações sensíveis e retorno de alertas ao usuário. A arquitetura contempla tanto o modo padrão de classificação durante a execução da navegação quanto o modo de captura de dados para treinamento e adaptação do modelo.

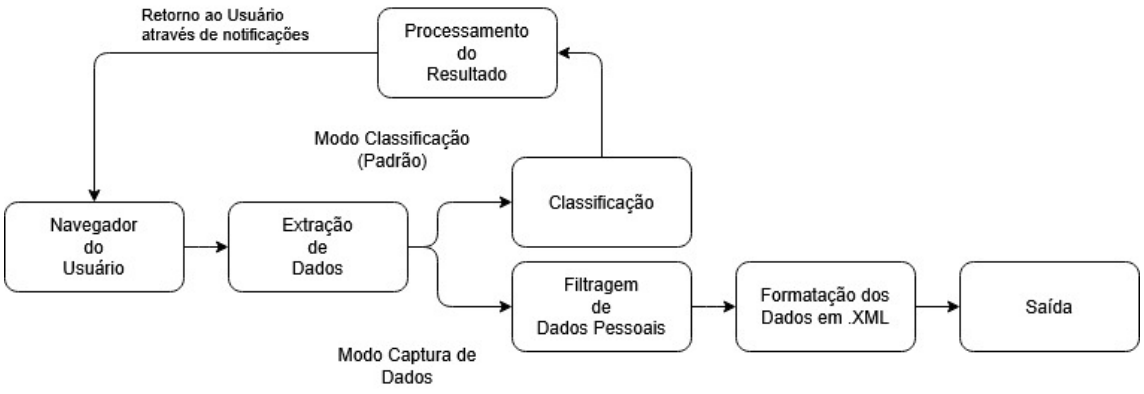

Figura 1. Arquitetura geral da ferramenta UAXD e fluxo de processamento dos dados.

A partir dessa arquitetura, o desempenho da ferramenta depende diretamente da qualidade e representatividade dos dados utilizados. Nesse contexto, torna-se fundamental a construção de um conjunto de dados capaz de refletir o tráfego real da Web e a diversidade de ataques XSS observados na prática, conforme descrito a seguir.

3.2. Conjunto de Dados Construído

Durante as etapas iniciais do desenvolvimento, foram realizados testes preliminares com dados públicos amplamente utilizados na literatura, com o objetivo de avaliar a

viabilidade inicial da abordagem proposta. Esses experimentos exploratórios utilizaram amostras benignas provenientes do *ECML/PKDD 2007 HTTP Dataset* e do *CSIC 2010 HTTP Dataset* e de amostras maliciosas geradas a partir da lista pública de *payloads* XSS de Taşdelen [İsmail Taşdelen, 2024]. Embora adequados para validações iniciais, esses dados apresentam baixa capacidade de generalização quando aplicados a tráfego real, evidenciando a necessidade da construção de uma base de dados própria e mais representativa, de modo a mitigar o *overfitting* frequentemente observado em estudos que utilizam bases artificiais simples, homogêneas ou pouco representativas dos cenários reais [Qin et al., 2024, Gallagher e Eliassi-Rad, 2009].

Diante desse cenário, optou-se pela construção de um conjunto de dados baseado em tráfego real, coletado diretamente a partir da navegação Web por meio da UAXD. A ferramenta foi equipada com um modo específico de captura de pacotes HTTP, permitindo a obtenção de amostras benignas provenientes de diferentes domínios, contextos e períodos de navegação.

Com o objetivo de aprimorar a capacidade de generalização do modelo, foi realizada uma captura abrangendo mais de 22.000 requisições legítimas. Sobre esse conjunto, aplicaram-se rigorosas etapas de pré-processamento, incluindo:

- **Normalização textual:** conversão de todo o conteúdo das requisições para letras minúsculas, mitigando variações irrelevantes.
- **Filtragem por tamanho:** remoção de requisições muito curtas (menores que 15 caracteres) ou excessivamente longas (maiores que 1000 caracteres), por serem majoritariamente irrelevantes ou ruidosas no contexto de ataques XSS.
- **Deduplicação:** remoção de amostras altamente semelhantes (85% ou mais de similaridade), evitando a sobre-representação de padrões específicos.

Após o pré-processamento, foram selecionadas 10.000 amostras benignas. A base maliciosa foi composta por 10.000 amostras geradas a partir de *payloads* públicos de Taşdelen [İsmail Taşdelen, 2024], contextualizados em diferentes tipos de requisições HTTP, incluindo métodos *GET* e *POST*, parâmetros de URL, corpo da requisição, estruturas JSON e cabeçalhos HTTP.

O conjunto final totaliza 20.000 amostras balanceadas entre *XSS* e *não-XSS*, garantindo aprendizado adequado das classes e evitando viés no treinamento. As requisições foram então transformadas em vetores numéricos por meio de técnicas de extração e representação de atributos, utilizados como entrada para os modelos de aprendizado de máquina.

Diferentemente de abordagens baseadas exclusivamente em dados públicos ou sintéticos, o dataset foi construído a partir de múltiplas fontes, combinando tráfego real de navegação com *payloads* públicos. Essa estratégia busca aumentar a variabilidade dos dados e reduzir limitações associadas a bases homogêneas ou pouco representativas, frequentemente observadas em datasets como o CSIC2010 [Lekies et al., 2013, Melicher et al., 2018, Melicher et al., 2021, Qin et al., 2024].

Apesar disso, reconhece-se que a natureza dinâmica da Web e o desbalanceamento típico de cenários reais ainda representam desafios para a completa generalização do modelo.

3.3. Modelo de Classificação e Ajustes para Tráfego Real

Esta subseção apresenta o modelo de classificação da UAXD e os ajustes para sua aplicação em tráfego real, incluindo as técnicas de extração de atributos, seleção do algoritmo, otimização de hiperparâmetros e estratégias para redução de falsos positivos.

3.3.1. Extração e Representação de Atributos

A eficácia dos modelos de detecção de XSS está diretamente relacionada à qualidade do processo de extração de atributos [Qin et al., 2024, Melicher et al., 2018, Melicher et al., 2021]. Trabalhos como Lekies *et al.* [Lekies et al., 2013] e Gallagher *et al.* [Gallagher e Eliassi-Rad, 2009] também reforçam que bases com atributos mal definidos tendem a gerar modelos altamente suscetíveis ao *overfitting* e com baixa capacidade de generalização. No entanto, o modelo inicial baseado apenas na presença ou ausência de determinados elementos (tags HTML, funções JavaScript, eventos) e métricas simples, como comprimento de carga útil e contagem de caracteres suspeitos apresenta desempenho limitado, especialmente na classificação de pacotes JSON, que naturalmente possuem alta ocorrência de caracteres como aspas, chaves e vírgulas.

Para superar essas limitações, são incorporadas heurísticas adicionais para diferenciar caracteres suspeitos, analisar o contexto de requisições JSON, aplicar métricas condicionais e considerar aspectos estruturais do HTML e de dados codificados, reduzindo falsos positivos em tráfego legítimo. Paralelamente, adota-se uma abordagem de vetorização textual para capturar padrões sintáticos e semânticos não identificáveis por expressões regulares. São utilizadas as técnicas *CountVectorizer* e *TF-IDF*, permitindo representar o conteúdo das requisições por meio de frequências e pesos estatísticos dos termos, de forma complementar aos atributos estruturais extraídos via *regex*. Além disso, diferentes granularidades foram testadas, incluindo:

- **N-gramas de caracteres (n=3 a 6):** especialmente eficazes na captura de padrões recorrentes de codificação, fragmentos de tags HTML e estruturas de *payloads* maliciosos.
- **N-gramas de palavras:** utilizados com menor efetividade, mas ainda úteis em certos contextos, especialmente em requisições JSON.

A estratégia final combina os atributos extraídos manualmente via *regex* com os vetores gerados pelo TF-IDF, utilizando o operador `FeatureUnion`. Este vetor combinado é então normalizado e utilizado como entrada para os modelos de aprendizado de máquina. Com isso, garante-se uma representação dos dados tanto sob a ótica estrutural (*regex*) quanto sob a ótica contextual e estatística (TF-IDF), promovendo uma redução significativa na taxa de falsos positivos e aumento da capacidade de detecção.

A partir dessa representação vetorial, procedeu-se à avaliação de diferentes algoritmos de classificação com o objetivo de identificar a abordagem mais adequada para a detecção de ataques XSS.

3.3.2. Avaliação de Modelos

Avaliam-se três algoritmos amplamente utilizados na detecção de XSS: *Random Forest*, *Decision Tree* e *AdaBoost*. A avaliação é baseada nas métricas de desempenho, priorizando modelos com maior capacidade de generalização e menor taxa de falsos positivos. Os modelos são treinados utilizando 70% dos dados disponíveis e validados nos 30% restantes do próprio conjunto de dados.

Os testes consideram um conjunto de hiperparâmetros extensivo, utilizando uma abordagem de *grid search*, conforme as combinações mostradas a seguir:

- **Random Forest**: parâmetros como *criterion* (*gini*, *entropy*), *n_estimators* (100, 200), *max_depth* (10, 20, 30), além de *min_samples_split*, *min_samples_leaf*, *max_features* e *bootstrap*.
- **Decision Tree**: ajustes similares, adicionando o parâmetro *splitter* (*best*, *random*).
- **AdaBoost**: explorando *n_estimators* (100, 200), *learning_rate* (0.5, 1.0, 1.5) e os algoritmos *SAMME* e *SAMME.R*.

Após múltiplas iterações, o **Random Forest** destacou-se significativamente, especialmente no equilíbrio entre acurácia e capacidade de generalização conforme observado na seção 4, corroborando os achados de trabalhos na literatura [Qin et al., 2024, Kulkarni e Lowe, 2016].

Com o modelo selecionado, procede-se à sua aplicação no ambiente de navegação real por meio da UAXD, permitindo a identificação de desafios práticos associados ao tráfego Web dinâmico, os quais motivam ajustes específicos para operação em produção.

3.3.3. Ajustes no Modelo para Tráfego Real

Ao submeter o modelo a testes em tempo de execução, utilizando a ferramenta UAXD, são identificados desafios específicos relacionados à natureza do tráfego real, sobretudo em pacotes que envolvem JSON e HTML dinâmico. Para mitigar esse problema e aprimorar a adaptação do modelo ao ambiente de produção, são adotadas duas estratégias complementares:

- **Ajuste de Confiabilidade**: limiar mínimo de 60% de confiança para que um alerta fosse emitido.
- **Implementação de *allowlist***: inclusão de domínios confiáveis a uma lista de exceções (*allowlist*) como `fonts.googleapis.com`, `youtube.com/youtubei/v1/log_event` e `play.google.com/log`.

Esses ajustes viabilizam a aplicação robusta do modelo em tráfego real. Na sequência, apresentam-se os componentes da UAXD responsáveis pela detecção e interação com o usuário.

3.4. Componentes da Ferramenta

Esta subseção descreve os principais componentes da UAXD, incluindo os módulos de monitoramento, sistema de alertas e configurações, evidenciando como esses elementos atuam de forma integrada na detecção de ataques XSS no ambiente do cliente.

3.4.1. Mecanismos de Monitoramento

A UAXD realiza o monitoramento contínuo de dados no lado do cliente por meio da interceptação de parâmetros de requisições HTTP e da observação de manipulações do DOM. Os parâmetros das requisições *GET* e *POST* são interceptados, normalizados, decodificados e enviados ao classificador, conforme implementado no módulo `utils.js` [Mozilla Foundation, 2025].

Paralelamente, o monitoramento de ataques do tipo *DOM-based* XSS é realizado por meio da sobrecarga de métodos do DOM que são responsáveis por modificar o conteúdo de elementos HTML. Sempre que uma dessas funções é executada (como alteração de atributos, inserção de elementos ou mudanças no conteúdo textual), o novo conteúdo é capturado e enviado para análise. Esse mecanismo é implementado através das funções `wrapSetter` e `wrapFunction` no módulo `domscanner.js`.

3.4.2. Sistema de Alertas

Quando uma requisição é classificada como potencialmente maliciosa, a UAXD gera um alerta visual para o usuário, bloqueando temporariamente a requisição e exibindo a fonte do pacote. A Figura 2 ilustra esse mecanismo.

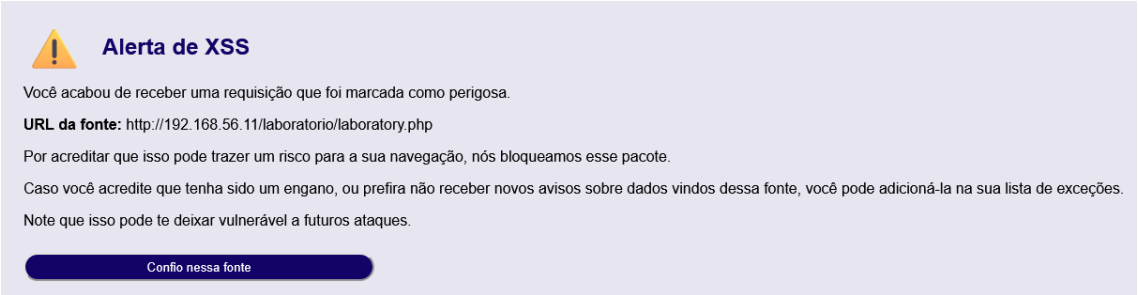

Figura 2. Tela de alerta de XSS exibida pela UAXD ao detectar uma requisição HTTP suspeita.

O usuário pode optar por confiar na fonte e adicioná-la à lista de exceções (*allowlist*), impedindo que alertas futuros sejam exibidos para requisições provenientes desse domínio.

3.4.3. Lista de Exceções e Configurações

A UAXD permite que o usuário gerencie uma lista de exceções, na qual domínios confiáveis podem ser adicionados manualmente ou através da própria interface de alerta. A Figura 3 mostra a tela de configurações da ferramenta, na qual a *whitelist* pode ser editada, exportada ou importada.

A UAXD também oferece um modo avançado de configurações (Figura 4), no qual é possível ajustar parâmetros como:

- **Confiabilidade mínima** para emissão de alertas (*threshold*) tanto para HTTP quanto para DOM, configurada por padrão em 60%.

- Ativação do **modo de treinamento**, permitindo a captura de tráfego para posterior geração de datasets personalizados.
- Controle sobre a coleta de dados analíticos.
- Opção de registrar pacotes mesmo de fontes na lista de exceções, útil para análises estatísticas.

A execução da UAXD apresentou baixo impacto no desempenho do navegador, com tempos médios de resposta na ordem de milissegundos, conforme observado durante a utilização da ferramenta (Figura 4). Isso indica viabilidade para uso contínuo sem prejuízo à experiência do usuário.

Figura 3. Configurações gerais da UAXD com lista de exceções.

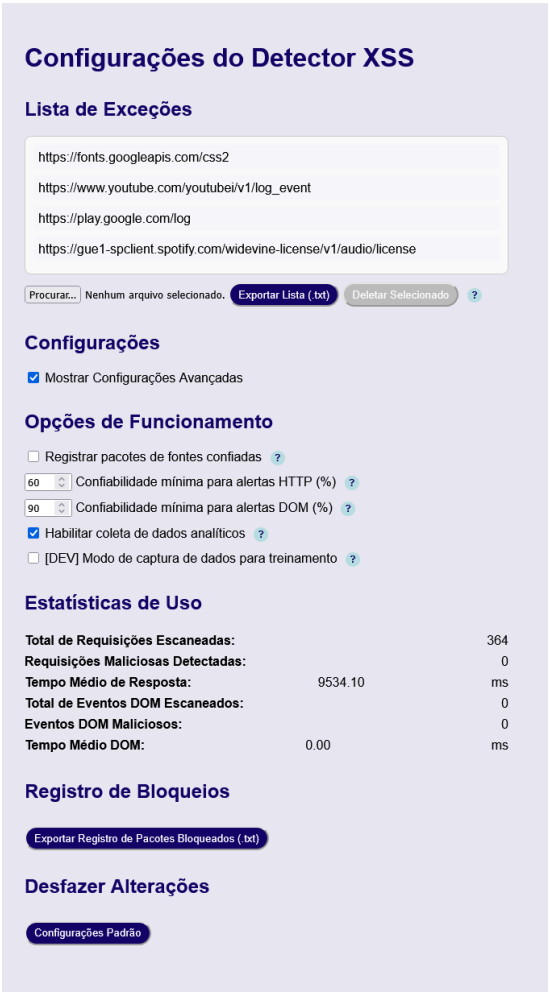

Figura 4. Configurações avançadas.

3.4.4. Modo de Treinamento e Gerenciamento de Dados

O modo de treinamento permite que a UAXD se adapte dinamicamente ao perfil de navegação do usuário final. O retreinamento do modelo é realizado a partir da captura contínua de tráfego de navegação do usuário, permitindo a incorporação de novos padrões de comportamento ao dataset. Esse processo ocorre de forma incremental, utilizando amostras coletadas localmente pela extensão, sem necessidade de infraestrutura externa.

Ao capturar tráfego real e possibilitar o retreinamento do modelo, a ferramenta reduz significativamente a incidência de falsos positivos, especialmente em cenários com aplicações Web específicas ou padrões de tráfego atípicos. Essa capacidade de personalização distingue a UAXD das soluções tradicionais descritas na literatura.

3.4.5. Interface de Popup e Registro de Bloqueios

A UAXD também conta com uma interface compacta acessível diretamente no ícone da barra do navegador (Figura 5), na qual o usuário pode verificar rapidamente o status da detecção, visualizar registros recentes de bloqueios e acessar as configurações.

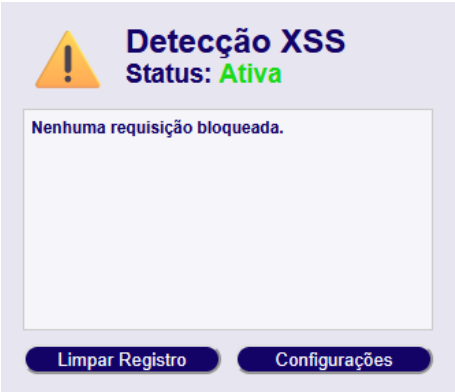

Figura 5. Interface popup da UAXD exibindo o status de detecção e registro de bloqueios.

O log de bloqueios da extensão armazena até 20 registros recentes, permitindo que o usuário visualize rapidamente as últimas detecções diretamente pela interface. De forma independente, o mecanismo de captura de pacotes utiliza um *buffer* temporário que mantém até 30 pacotes na memória local antes de transferi-los para a base de dados persistente no *IndexedDB*. O tamanho do *buffer* foi definido empiricamente, buscando equilíbrio entre consumo de memória e capacidade de retenção de dados para treinamento.

Com esses componentes, a UAXD integra mecanismos de monitoramento, classificação e interação com o usuário em uma única ferramenta voltada à detecção de ataques XSS no lado do cliente. A seguir, são apresentados os resultados experimentais que avaliam o desempenho do modelo de classificação e o impacto das estratégias de extração de atributos adotadas.

4. Resultados

Esta seção apresenta a avaliação experimental da UAXD, analisando o impacto das estratégias de extração de atributos e o desempenho dos modelos, com base em validação cruzada e testes em tempo real, evidenciando sua eficácia na detecção de ataques XSS.

4.1. Critérios utilizados

São empregadas técnicas de vetorização e engenharia de atributos, nas quais o extrator transforma requisições HTTP em características numéricas utilizadas pelos modelos de aprendizado de máquina. Diferentes versões desse extrator foram avaliadas ao longo

do desenvolvimento, e a Figura 6 apresenta o impacto dessas variações no desempenho do classificador, medido por Acurácia, Precisão, Recall e F1-Score.

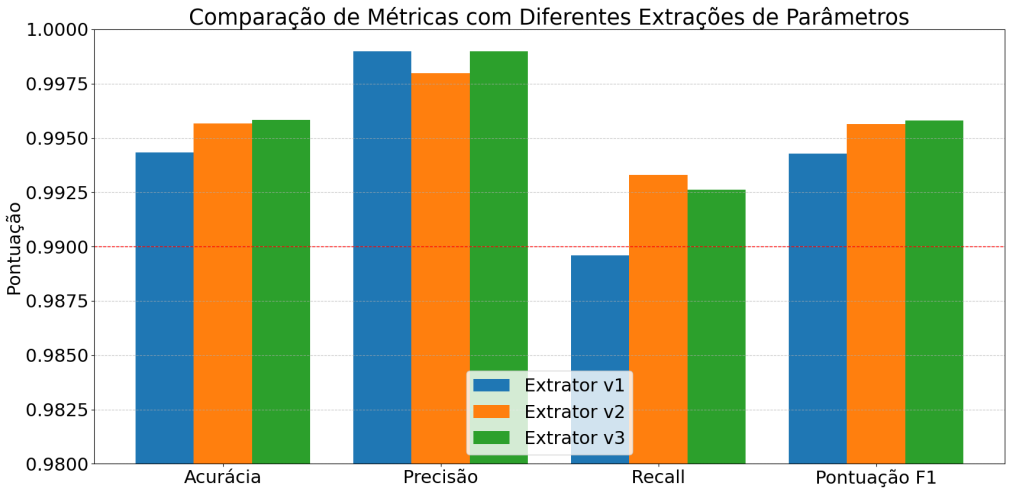

Figura 6. Comparação de métricas entre diferentes versões do extrator de atributos.

Além da evolução manual do extrator de atributos baseado em expressões regulares, foi implementada e testada uma abordagem híbrida que combina técnicas baseadas em *regex* com vetorização textual. Essa abordagem utiliza modelos de vetorização como *CountVectorizer* e *TF-IDF*, capazes de capturar padrões mais sutis e relações semânticas entre os elementos das requisições.

A Figura 7 apresenta os resultados comparativos entre três estratégias: uso exclusivo de expressões regulares (*Regex*), uso apenas do vetorizador textual e a abordagem combinada, que se mostrou superior em praticamente todas as métricas avaliadas.

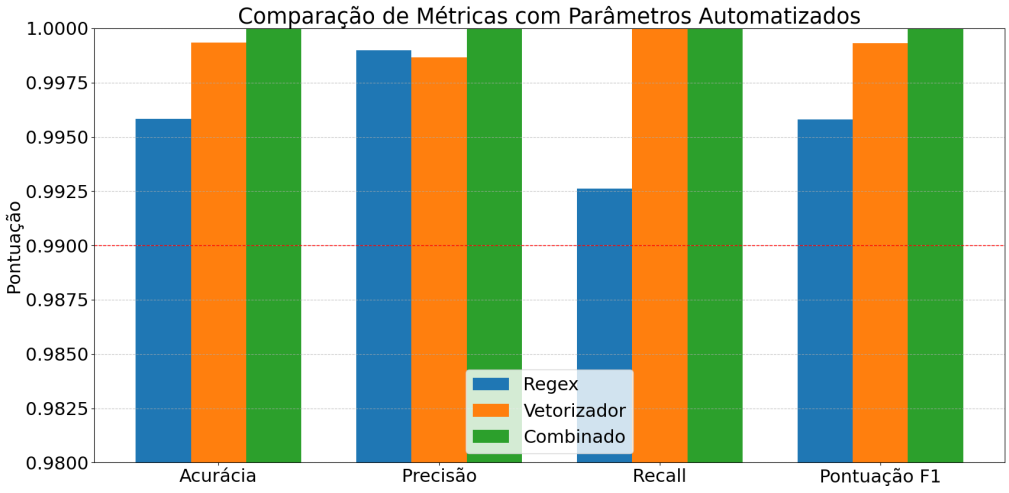

Figura 7. Comparação de desempenho entre diferentes métodos de extração de atributos: Regex, Vetorizador e abordagem combinada.

Observa-se que a combinação de métodos supera significativamente as abordagens isoladas, uma vez que permite capturar tanto características estruturais (via *regex*)

quanto padrões textuais e contextuais (via vetorização). Este resultado está em consonância com achados de trabalhos recentes, como [Qin et al., 2024], que apontam que métodos híbridos são particularmente eficazes na detecção de XSS.

4.2. Descrição dos Resultados

Além dos testes de laboratório realizados por meio de validação cruzada estratificada, a avaliação do classificador foi complementada por testes em tempo real, utilizando a ferramenta UAXD durante sessões de navegação em múltiplos domínios populares, envolvendo diferentes tipos de aplicações Web (e.g., redes sociais, streaming e serviços online), permitindo avaliar o comportamento do modelo em cenários heterogêneos e sua capacidade de generalização frente ao tráfego real e dinâmico da Web.

O desempenho do modelo foi avaliado por meio de validação cruzada estratificada com cinco divisões. A Figura 8 e a Tabela 1 apresentam os resultados, destacando-se que, na maioria dos cenários, as métricas de *accuracy*, *precision* e *F1-Score* mantiveram-se consistentemente acima de 0,99 — evidenciado pela linha vermelha tracejada.

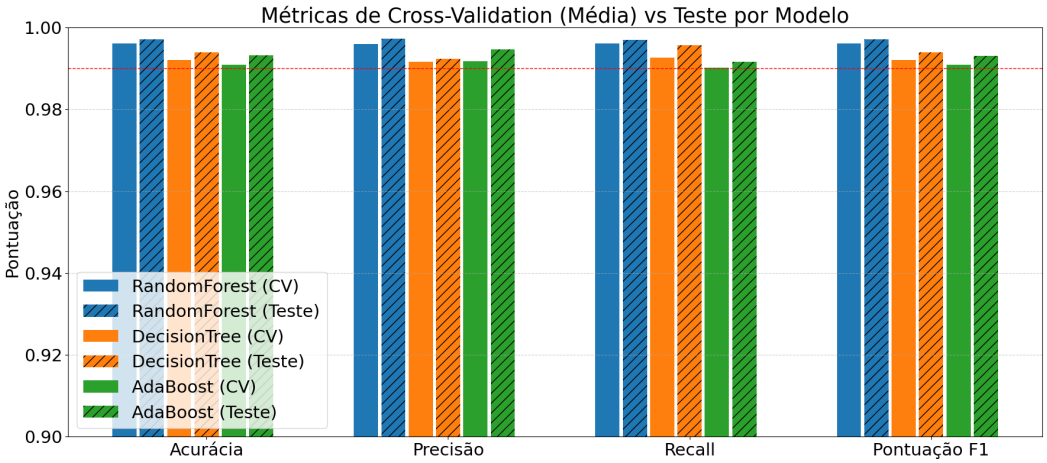

Figura 8. Comparação de desempenho entre validação cruzada e teste.

Tabela 1. Desempenho dos modelos na validação cruzada e no conjunto de teste

Modelo	Conjunto	Acurácia	Precisão	Recall	F1-Score
Random Forest	Média Cross Validation	0.9961	0.9960	0.9962	0.9961
	Teste	0.9972	0.9973	0.9970	0.9972
Decision Tree	Média Cross Validation	0.9921	0.9916	0.9926	0.9921
	Teste	0.9940	0.9923	0.9956	0.9940
AdaBoost	Média Cross Validation	0.9909	0.9917	0.9902	0.9909
	Teste	0.9932	0.9946	0.9916	0.9931

Embora as diferenças entre os modelos avaliados sejam pequenas, o *Random Forest* apresentou maior estabilidade e melhor equilíbrio entre precisão e *recall*, justificando sua escolha como modelo final, em linha com resultados reportados na literatura [Qin et al., 2024, Kulkarni e Lowe, 2016, Prasetio et al., 2021].

Apesar das métricas superiores a 99%, é importante considerar que o conjunto de dados foi parcialmente construído em ambiente controlado. Em cenários reais, caracteri-

zados por maior complexidade e desbalanceamento, o desempenho pode variar, de modo que os resultados devem ser interpretados como indicativo de viabilidade da abordagem.

5. Conclusão e Trabalhos Futuros

Os resultados obtidos indicam que a UAXD apresenta um bom equilíbrio entre desempenho, robustez e aplicabilidade, superando limitações observadas em trabalhos relacionados. O modelo demonstra elevada eficácia na detecção de ataques XSS refletidos, armazenados e *DOM-based* em ambiente real de navegação, aliado à integração no navegador, que permite resposta imediata e adaptação ao perfil do usuário. Destaca-se, ainda, o modo de treinamento, que possibilita o retreinamento contínuo do classificador, ampliando sua capacidade de adaptação a diferentes cenários.

Apesar dos avanços, algumas limitações apontam oportunidades de evolução. O monitoramento atual do DOM é eficaz para injeções simples, mas apresenta restrições frente a ataques mais sofisticados, motivando investigações futuras em técnicas de análise semântica de HTML e *scripts* dinâmicos. Da mesma forma, embora as heurísticas adotadas reduzam falsos positivos em tráfego JSON, há espaço para o desenvolvimento de modelos especializados para dados estruturados.

Outro aspecto relevante refere-se à portabilidade da UAXD. Embora implementada inicialmente no Mozilla Firefox, a adoção do padrão WebExtensions possibilita sua adaptação para outros navegadores, ampliando seu alcance. Como trabalho futuro, pretende-se expandir o dataset com dados coletados de diferentes usuários e avaliar o modelo em datasets públicos amplamente utilizados, permitindo maior diversidade de cenários e comparação com a literatura.

Em síntese, ao priorizar a execução no lado do cliente, com foco em ataques XSS baseados em DOM e suporte a retreinamento adaptativo, a UAXD se diferencia das abordagens tradicionais e se mostra uma solução promissora para a proteção direta do usuário final.

Agradecimentos

Este trabalho é parcialmente financiado pelo CNPq, CAPES, FAPERJ, RNP, CEFET/RJ e faz parte do INCT de Redes de Comunicação e Internet das Coisas Inteligentes (ICoNIoT), financiado pelo CNPq (405940/2022-0) e pela CAPES (88887.954253/2024-00).

Os autores usaram a ferramenta de inteligência artificial generativa ChatGPT exclusivamente para correção ortográfica e gramatical do texto e auxílio técnico à escrita de código e à construção do conjunto de dados maliciosos a partir de dados reais, não substituindo a autoria ou a análise crítica dos resultados pelos autores.

Referências

Bastos, I. V., Guarizi, B. D., Alves, I. M. M., Souza, J. A. F. e., Pimentel, G. O., Watanabe, J. A. C., Mascarenhas, D. M., Rubinstein, M. G. e Moraes, I. M. (2025). Cross-site scripting: Ataques, detecção e contramedidas. Em *Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*. SBC - Sociedade Brasileira de Computação. Minicurso apresentado no SBRC 2025.

Gallagher, B. e Eliassi-Rad, T. (2009). Classification of http attacks: a study on the ecml/pkdd 2007 discovery challenge. Relatório técnico, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

Kulkarni, A. D. e Lowe, B. (2016). Random forest algorithm for land cover classification.

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.

Lekies, S., Stock, B. e Johns, M. (2013). 25 million flows later: large-scale detection of dom-based xss. Em *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, p. 1193–1204.

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.

Melicher, W., Das, A., Sharif, M., Bauer, L. e Jia, L. (2018). Riding out domsday: Towards detecting and preventing dom cross-site scripting. Em *2018 Network and Distributed System Security Symposium (NDSS)*.

Melicher, W., Fung, C., Bauer, L. e Jia, L. (2021). Towards a lightweight, hybrid approach for detecting dom xss vulnerabilities with machine learning. Em *Proceedings of the Web Conference 2021*, p. 2684–2695.

Mozilla Foundation (2025). Módulos JavaScript. Accessed: 2025-06-06.

Oshoiribhor, E. e John-Otumu, A. (2025). Xss-net: An intelligent machine learning model for detecting cross-site scripting (xss) attack in web application. *Mach. Learn. Res*, 10:14–24.

OWASP Foundation (2021a). OWASP Top 10 - 2021: The Ten Most Critical Web Application Security Risks. Accessed: 2025-06-10.

OWASP Foundation (2021b). OWASP Top 10 - 2025: The Ten Most Critical Web Application Security Risks. Accessed: 2026-02-03.

Prasetio, D. A., Kusrini, K. e Arief, M. R. (2021). Cross-site scripting attack detection using machine learning with hybrid features. *Jurnal Infotel*, 13(1):1–6.

Qin, Q., Li, Y., Mi, Y., Shen, J., Wu, K. e Wang, Z. (2024). Detecting xss with random forest and multi-channel feature extraction. *Computers, Materials & Continua*, 80(1).

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.

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

İsmail Taşdelen (2024). Cross site scripting (xss) vulnerability payload list. `https://github.com/payloadbox/xss-payload-list`. Acesso em: 06 maio 2025.