WebTrap: Uma ferramenta de detecção de ataques em requisições HTTP utilizando Aprendizado de Máquina

Pedro Henrique Ferreira Wendling[1], **Dalbert Matos Mascarenhas**[1]

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet/RJ)
Petrópolis, RJ, Brasil

pedro.wendling@aluno.cefet-rj.br, dalbert.mascarenhas@cefet-rj.br

Abstract. *According to the OWASP Top 10 2025 report, Injection vulnerabilities remain among the most critical security risks for modern web applications. This work proposes a web honeypot that integrates two machine learning models with deception mechanisms. The first model identifies HTTP requests associated with SQL Injection, Cross-Site Scripting, Remote Code Execution, and Local File Inclusion attacks, distinguishing them from benign traffic, while the second model classifies the detected attack category. The results achieved detection rates above 94% for malicious traffic generated by tools such as sqlmap, XSSER, Commix, and Wfuzz. For benign requests, the proposed approach achieved an accuracy of approximately 93%, demonstrating the viability of the proposed approach for intelligent web honeypots.*

Resumo. *Segundo o relatório OWASP Top 10 2025, vulnerabilidades do tipo Injection permanecem entre os principais riscos de segurança para aplicações web. Este trabalho propõe um honeypot web que integra dois modelos de aprendizado de máquina a mecanismos de deception. O primeiro modelo identifica requisições HTTP associadas aos ataques SQL Injection, Cross-Site Scripting, Remote Code Execution e Local File Inclusion, distinguindo-as do tráfego benigno, enquanto o segundo classifica a categoria do ataque detectado. Os resultados indicaram taxas de detecção superiores a 94% para tráfego malicioso gerado por ferramentas como sqlmap, XSSER, Commix e Wfuzz. Para requisições benignas, foi obtida uma taxa de acerto de aproximadamente 93%, evidenciando a viabilidade da abordagem proposta para honeypots web inteligentes.*

1. Introdução

Com o crescimento da utilização de aplicações web em serviços governamentais, financeiros e comerciais, ataques direcionados a sistemas HTTP tornaram-se uma das principais ameaças à segurança da informação [Noman et al. 2020, McNally and Curran 2024, Kaya et al. 2026]. A sofisticação, velocidade e escala desses ataques aumentaram significativamente nos últimos anos, impulsionadas, em muitos casos, pelo uso crescente de técnicas baseadas em Inteligência Artificial por agentes maliciosos para automatizar processos de reconhecimento, exploração e evasão de mecanismos de defesa [Palo Alto Networks , IBM]. Esse cenário amplia os riscos relacionados à confidencialidade, integridade e disponibilidade das informações, possibilitando desde vazamento de dados até comprometimento completo de aplicações web [Sharif 2022].

Segundo o relatório OWASP Top 10 de 2025, vulnerabilidades da categoria *Injection* permanecem entre os riscos mais críticos e frequentemente explorados em aplicações

web modernas [OWASP Foundation]. Ataques como SQL Injection (SQLi), Cross-Site Scripting (XSS), Remote Code Execution (RCE) e Local File Inclusion (LFI) continuam sendo amplamente utilizados devido ao seu potencial de exploração e impacto sobre sistemas vulneráveis [Faisal Fadlalla and Elshoush 2023].

Tradicionalmente, Sistemas de Detecção de Intrusão (IDS) utilizam abordagens baseadas em assinaturas estáticas e padrões previamente conhecidos [Buczak and Guven 2016]. Embora eficazes contra ameaças catalogadas, esses sistemas apresentam limitações diante de técnicas modernas de evasão, payloads ofuscados e ataques do tipo *zero-day*, reduzindo sua capacidade de adaptação a novos cenários de ameaça [Zhang et al. 2008, Farzaan et al. 2024, Rahman 2025]. Nesse contexto, técnicas de aprendizado de máquina vêm sendo exploradas como alternativa para aprimorar mecanismos de detecção em sistemas de informação, devido à sua capacidade de generalização e identificação de padrões comportamentais em tempo real [Nimmagadda and Mehr 2025, Sen et al. 2022, Guarizi et al. 2025].

Paralelamente, honeypots vêm sendo utilizados como mecanismos de *deception* para monitoramento de ameaças e análise do comportamento de atacantes [López et al. 2024, Sayari and Rekhis 2025]. Diferentemente de mecanismos tradicionais de defesa, honeypots possuem como objetivo atrair interações maliciosas, coletando informações relevantes sobre técnicas de exploração utilizadas em ataques reais [Zhang et al. 2003].

Diante desse cenário, este trabalho propõe o desenvolvimento de uma ferramenta que integra um honeypot web implementado em Flask a dois modelos de aprendizado de máquina baseados no algoritmo Random Forest: um modelo de detecção, responsável por identificar requisições HTTP associadas a ataques de SQL Injection, Cross-Site Scripting, Remote Code Execution e Local File Inclusion, distinguindo-as do tráfego benigno; e um modelo de classificação de ataques, que rotula o tipo de ataque identificado. Ambos foram treinados utilizando um dataset próprio composto por 39.525 requisições HTTP, incluindo tráfego legítimo e ataques contendo diferentes técnicas de evasão e ofuscação, sendo que o segundo utilizou apenas as requisições maliciosas do conjunto de dados.

Além da capacidade de detecção, a ferramenta proposta incorpora mecanismos de *deception*, permitindo retornar respostas distintas para diferentes comportamentos identificados nas requisições analisadas. Dessa forma, a ferramenta busca unir mecanismos de monitoramento, detecção, classificação e interação com atacantes em uma única plataforma experimental.

As principais contribuições desse estudo são:

1. Avaliação das limitações de datasets públicos de requisições HTTP para treinamento de modelos de aprendizado de máquina voltados à detecção de ataques web.
2. Criação de um conjunto de dados de requisições web, contendo tráfegos benignos e malignos, gerados artificialmente e capturados via navegação automatizada.
3. Implementação de um honeypot web inteligente em um servidor Flask integrando aprendizado de máquina e mecanismos de *deception*.

O restante deste trabalho está organizado da seguinte forma: A Seção 2 descreve e discute os trabalhos relacionados. A Seção 3 descreve as ferramentas implementadas e

o desenvolvimento da solução. A Seção 4 discute os resultados obtidos. A seção 5 reflete sobre os resultados e limitações da ferramenta. Enquanto a Seção 6 conclui este trabalho e descreve objetivos futuros.

2. Trabalhos Relacionados

O artigo de [Thang 2020] trata do uso de algoritmos de Random Forest na detecção de ataques web, utilizando o dataset CSIC 2010, por meio da análise de atributos extraídos de requisições HTTP. O modelo desenvolvido apresentou uma métrica de recall de 98,76% e uma taxa de falsos positivos em 3,2%, obtendo desempenho superior ao de outras abordagens comparadas no estudo, indicando a boa qualidade do uso de Random Forest nesse contexto.

A pesquisa de [Rong et al. 2018] trata do uso de um modelo de redes neurais de convolução (CNN) para detecção de ataques web do tipo *injection*, incluindo uma capacidade de adaptação contínua e análise da relação entre os caracteres da string. O modelo foi treinado utilizando um dataset com 4302 amostras, obtendo uma taxa de falso positivos de 0,02% e ressaltando a viabilidade do uso de aprendizado de máquina na detecção de tráfegos web maliciosos, bem como a importância de se considerar os caracteres textuais como features.

No estudo apresentado por [Komiya et al. 2011], é proposta a utilização de técnicas de aprendizado de máquina para detecção de ataques web do tipo SQL Injection e Cross-Site Scripting, utilizando extração de features e tokenização das requisições. Os resultados demonstraram elevada eficácia do modelo SVM com kernel Gaussiano (RBF), alcançando precisão de 99,16% para SQL Injection e 98,95% para ataques XSS.

Em [Nimmagadda and Mehr 2025], é desenvolvido um sistema inteligente de detecção de intrusão integrado a um honeypot SSH e Telnet, utilizando um modelo baseado em Random Forest para identificação de ataques, incluindo ameaças do tipo *zero-day*. Os autores obtiveram resultados promissores, com acurácia superior a 95% na detecção de ataques conhecidos e baixo tempo de predição, indicando a viabilidade da combinação dessas duas tecnologias.

Diferentemente dos trabalhos relacionados, o *WebTrap* integra um honeypot web a dois modelos de aprendizado de máquina voltados à análise de parâmetros HTTP: um detector de requisições maliciosas associadas a ataques web e outro classificador do tipo de ataque. Além disso, a ferramenta incorpora mecanismos de *deception*, capazes de retornar respostas distintas de acordo com o comportamento identificado nas requisições maliciosas. Ambos os modelos utilizam o algoritmo Random Forest, abordagem que apresentou elevado desempenho em estudos anteriores, como observado em [Correia and Pedrini 2020]. Dessa forma, o *WebTrap* reúne, em uma única plataforma, mecanismos de detecção, classificação, coleta de informações e interação com atacantes.

3. A Ferramenta Proposta *WebTrap*

O *WebTrap* integra um honeypot web desenvolvido em Flask a dois modelos de aprendizado de máquina: um modelo de detecção, responsável por classificar requisições HTTP como benignas ou maliciosas, e um modelo de classificação de ataques, utilizado para identificar o tipo do ataque detectado. O honeypot simula um portal governamental contendo páginas de autenticação e busca, registra todas as requisições em arquivos

NDJSON e emprega mecanismos de *deception* para retornar respostas dinâmicas às requisições maliciosas.

3.1. Tecnologias Implementadas

A ferramenta foi desenvolvida em Python. Os modelos de aprendizado de máquina foram implementados com Scikit-learn, enquanto o honeypot web foi desenvolvido utilizando Flask. As páginas da aplicação foram implementadas em HTML e a coleta automatizada de requisições HTTP empregou Selenium e bibliotecas auxiliares para geração e captura de tráfego.

A Figura 1 apresenta a arquitetura geral da ferramenta proposta.

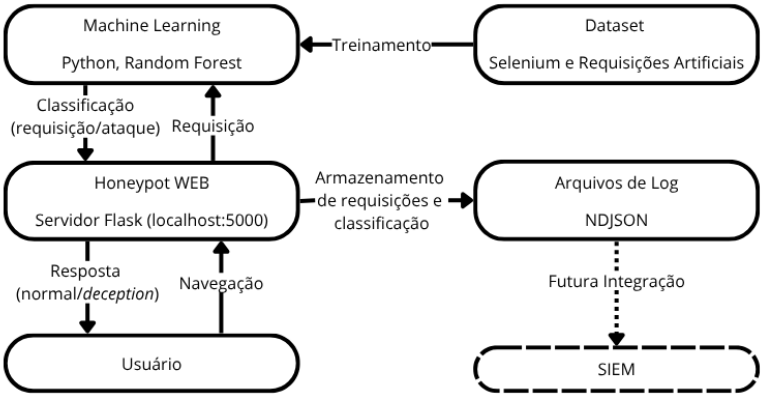

Figura 1. Fluxograma dos componentes do *WebTrap*.

3.2. Dataset e Treinamento dos Modelos

Inicialmente, foram analisados datasets públicos contendo tráfego web benigno e malicioso para utilização no treinamento do modelo detector de ataques. Entre os conjuntos de dados considerados destacam-se o CIC IDS 2017, desenvolvido pelo *Canadian Institute for Cybersecurity* (CIC), além dos datasets ECML/PKDD 2007 Discovery Challenge, desenvolvido pela *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (ECML/PKDD), e CSIC 2010, desenvolvido pelo *Consejo Superior de Investigaciones Científicas* (CSIC). O dataset CIC IDS 2017 foi obtido diretamente a partir da fonte oficial disponibilizada em [Canadian Institute for Cybersecurity], enquanto os datasets ECML/PKDD 2007 e CSIC 2010 foram obtidos por meio de repositórios públicos no GitHub, referenciados em [Matthew Sudol].

Embora os datasets analisados apresentassem informações relevantes sobre tráfego benigno e malicioso, observou-se que eles não eram totalmente adequados à proposta deste trabalho, considerando que a ferramenta desenvolvida realiza a análise exclusivamente a partir do método HTTP e dos parâmetros presentes nas requisições. Os principais desafios identificados foram:

1. Baixa diversidade de requisições benignas, limitando a capacidade de generalização do modelo e aumentando a ocorrência de *overfitting* nos testes.
2. Presença significativa de requisições artificiais compostas por sequências textuais aleatórias de baixa correlação com tráfego legítimo real, comprometendo a capacidade do modelo em distinguir adequadamente requisições benignas e maliciosas.

3. Predominância de payloads maliciosos altamente ofuscados ou utilizando técnicas avançadas de evasão, com pouca representatividade de ataques simples, dificultando o aprendizado de padrões básicos relacionados aos ataques web analisados.

Após a realização de testes experimentais utilizando esses datasets, concluiu-se que a adaptação e o tratamento individual dos problemas identificados demandariam elevado esforço de tratamento e adaptação. Dessa forma, optou-se pela criação de um dataset próprio, permitindo maior controle sobre a composição, balanceamento e diversidade das requisições utilizadas no treinamento dos modelo de detecção empregado pelo *WebTrap*.

3.2.1. Composição do Dataset de Treinamento

O dataset desenvolvido foi composto por 39.525 requisições web contendo classificação, método HTTP e parâmetros, armazenados em formato CSV. Aproximadamente 55% das amostras correspondem a tráfego benigno, enquanto os 45% restantes representam tráfego malicioso, distribuído igualmente entre os ataques abordados neste trabalho, incluindo variantes ofuscadas.

As requisições benignas foram obtidas tanto por geração artificial baseada em padrões observados em aplicações web reais quanto por navegação automatizada utilizando Selenium. Já as amostras maliciosas foram geradas a partir das mesmas estruturas utilizadas no tráfego benigno, porém contendo payloads específicos para cada categoria de ataque, além de amostras obtidas por meio da ferramenta *sqlmap* para ataques SQL Injection.

A coleta de tráfego benigno utilizando Selenium foi realizada de forma controlada, respeitando as diretivas definidas no arquivo *robots.txt* e utilizando intervalos curtos entre as requisições. Foram utilizados domínios de *e-commerce* e enciclopédia devido à presença de URLs contendo parâmetros diversificados. Após a coleta, os dados passaram por etapas de normalização textual e remoção de termos específicos dos domínios acessados.

Para geração das requisições artificiais, foram utilizadas quatro estruturas principais:

1. Simulação de buscas utilizando o método GET, inspiradas em mecanismos de pesquisa e compostas por parâmetros como *search*, *id*, *page*, *action*, *client* e *sort*;
2. Simulação de buscas em fóruns utilizando o método GET, inspiradas em plataformas de fóruns e compostas por parâmetros como *q*, *type*, *sort* e *t*;
3. Simulação de autenticação utilizando o método POST, baseada em parâmetros como *username*, *password*, *remember* e *csrf_token*;
4. Simulação de formulários do tipo POST relacionados a autenticação, cadastro de usuários, busca de produtos, formulários de contato, avaliações e pagamentos.

As requisições GET artificiais foram geradas por scripts em Python, nos quais os valores dos parâmetros eram selecionados aleatoriamente a partir de um dicionário desenvolvido para o projeto. Além disso, cada amostra possuía probabilidade de sofrer alterações estruturais, como remoção ou modificação de parâmetros, aumentando a diversidade do dataset e reduzindo problemas relacionados a *overfitting*.

O dicionário utilizado contém 444 palavras frequentemente observadas em contextos legítimos de navegação web, incluindo termos em português e inglês relacionados a produtos, objetos e serviços. Parte dessas palavras foi selecionada propositalmente por apresentar sobreposição com termos frequentemente encontrados em ataques web, como "root", utilizado em expressões legítimas como "rooter wifi".

Para geração das requisições POST relacionadas a autenticação, foram utilizados nomes de usuário provenientes do arquivo *top-usernames-shortlist.txt* e senhas retiradas do arquivo *rockyou.txt*, ambos pertencentes às SecLists, disponível públicamente em [danielmiessler].

Além disso, visando aumentar a capacidade de generalização do modelo para requisições POST, foi desenvolvido um servidor Flask contendo formulários web preenchidos automaticamente por meio do Selenium utilizando dados coerentes com o contexto de cada formulário. As requisições geradas durante essas interações foram posteriormente incorporadas ao dataset de treinamento.

As amostras maliciosas foram geradas utilizando as mesmas estruturas aplicadas ao tráfego benigno, porém contendo payloads específicos para cada categoria de ataque. Parte dessas amostras foi composta exclusivamente pelos payloads maliciosos, enquanto outras combinavam termos legítimos do dicionário desenvolvido com os payloads correspondentes. Aproximadamente 30% das amostras maliciosas de cada ataque foram submetidas a uma ou duas camadas de codificação, simulando técnicas de evasão frequentemente observadas em ataques reais.

Os payloads utilizados neste trabalho foram obtidos a partir de repositórios públicos especializados em segurança ofensiva. Os payloads referentes a SQL Injection, Cross-Site Scripting e Local File Inclusion foram retirados das SecLists, enquanto os relacionados a Remote Code Execution foram obtidos no repositório PayloadsAllTheThings, ambos disponíveis publicamente em [danielmiessler , swisskyrepo].

3.2.2. Treinamento dos Modelos

O treinamento dos modelos foi realizado em Python utilizando as bibliotecas Scikit-learn, para implementação do algoritmo Random Forest, Pandas, para manipulação do dataset, SciPy, para integração das features numéricas e textuais, e Joblib, para persistência dos modelos treinados e arquivos auxiliares. O algoritmo Random Forest foi escolhido por apresentar elevado desempenho na detecção de ataques web, conforme observado em estudos anteriores [Correia and Pedrini 2020, Thang 2020]. O modelo de detecção foi treinado utilizando o dataset completo, enquanto o modelo de classificação de ataques utilizou apenas as requisições rotuladas como maliciosas.

Os hiperparâmetros do Random Forest foram definidos após diversos experimentos visando maximizar o desempenho do modelo e reduzir problemas de *overfitting*, resultando na configuração apresentada a seguir:

- n_estimators = 200;
- max_depth = 20;
- min_samples_leaf = 2;
- class_weight = "balanced";

- random_state = 42;

Como forma de comparação, também foram realizados experimentos com o algoritmo XGBoost, empregando hiperparâmetros equivalentes aos utilizados no Random Forest. Os resultados obtidos são apresentados na Seção 4.

- n_estimators = 200;
- max_depth = 20;
- learning_rate=0.1;
- random_state = 42;

Ambos os modelos utilizam exatamente os mesmos três grupos de features: numéricas, textuais e relacionadas ao método HTTP, diferindo apenas no conjunto de dados utilizado e objetivo. As features numéricas representam características estruturais das requisições; as textuais capturam padrões de caracteres frequentemente presentes em payloads maliciosos; e as relacionadas ao método HTTP distinguem características específicas entre requisições GET e POST.

As features numéricas utilizadas são apresentadas a seguir:

- **Tamanho da Requisição (F1):** quantidade total de caracteres presentes na requisição. Payloads que empregam técnicas de evasão ou codificação tendem a aumentar o tamanho das requisições [Thang 2020].
- **Quantidade de Parâmetros (F2):** contagem de ocorrências do caractere “=”, comum aparecerem em grandes quantidades em payloads maliciosos.
- **Quantidade de Caracteres “+” (F3):** contagem do caractere “+”, frequentemente utilizado em codificações URL e técnicas de evasão.
- **Quantidade de Apóstrofes (F4):** contagem do caractere “’”, amplamente presente em payloads de SQL Injection.
- **Quantidade de Caracteres Especiais (F5):** contagem de caracteres incomuns em tráfego benigno, como “<”, “>”, “;”, “\” e aspas, frequentes em payloads de ataques web.
- **Quantidade de Caracteres Codificados (F6):** contagem de sequências iniciadas por "%"em codificação URL, geralmente associados a técnicas de evasão e ofuscação de payloads, como "%2F"para "/".
- **Quantidade de Trechos de Traversal (F7):** contagem de padrões referentes à navegação entre diretórios em sistemas Windows e Linux, incluindo variantes codificadas, característica de Local File Inclusion.
- **Quantidade de Termos Relacionados a SQL Injection (F8):** contagem de palavras associadas à linguagem SQL, frequentemente utilizadas em ataques SQL Injection. Optou-se por utilizar uma feature quantitativa em vez de binária devido à possibilidade de sobreposição desses termos em contextos benignos.
- **Quantidade de Comentários SQL (F9):** contagem de padrões de comentários SQL, como “–” e “/**/”, incomuns e utilizados em payloads de SQL Injection.
- **Quantidade de Termos Relacionados a XSS (F10):** contagem de palavras e padrões frequentemente associados a ataques Cross-Site Scripting.
- **Quantidade de Operadores de RCE (F11):** contagem de operadores utilizados em ataques de Remote Code Execution, como “;”, “|”, “&&” e “$(”.

- **Quantidade de Funções Relacionadas a RCE (F12):** contagem de termos associados a funções frequentemente exploradas em ataques de execução remota, como “exec”, “eval” e “system”.
- **Quantidade de Termos Específicos de RCE (F13):** contagem de comandos e palavras frequentemente presentes em payloads RCE, como “cmd”, “command” e “ping”.

A utilização conjunta dessas features teve como objetivo aumentar a capacidade do modelo de detecção em diferenciar padrões presentes em tráfego benigno e malicioso, já para o modelo classificador em identificar características típicas dos ataques abordados. Para ataques de Remote Code Execution, foram desenvolvidas múltiplas features especializadas devido à elevada diversidade estrutural observada nesse tipo de payload.

Experimentos preliminares mostraram que o uso exclusivo de features numéricas era insuficiente para representar padrões associados a técnicas de evasão e ofuscação, além de possuir uma generalização precária. Dessa forma, foi empregado o *TfidfVectorizer* da biblioteca Scikit-learn para extração de n-gramas de caracteres, utilizando a configuração a seguir:

- analyzer = "char";
- ngram_range = (3, 5);
- max_features = 1500;
- min_df = 3;
- max_df = 0.85;
- sublinear_tf = True;

A utilização de n-gramas de caracteres permitiu capturar padrões frequentemente presentes em payloads maliciosos, incluindo caracteres codificados, operadores de evasão e fragmentos parciais de comandos, reduzindo a dependência exclusiva de palavras completas.

O último grupo de features utilizado corresponde ao método HTTP das requisições, permitindo ao modelo distinguir características específicas entre tráfego GET e POST.

As features numéricas, textuais e categóricas foram integradas utilizando as funções *csr_matrix* e *hstack* da biblioteca SciPy. As features numéricas foram normalizadas antes do treinamento por meio de um *scaler*. Em ambos os modelos, os dados foram particionados em 80% para treinamento e 20% para teste.

3.3. Honeypot Web

A ferramenta proposta integra um servidor web desenvolvido em Flask a dois modelos de aprendizado de máquina, responsáveis pela detecção de requisições maliciosas e pela classificação do tipo de ataque. Durante a interação dos usuários com a plataforma, as requisições HTTP são capturadas, analisadas pelos modelos de aprendizado de máquina e respondidas de acordo com o comportamento identificado.

O servidor web é disponibilizado por meio do endereço IP da máquina hospedeira na porta 5000, simulando um portal governamental. A interface da aplicação foi estruturada em três diretórios principais: página inicial, página de autenticação administrativa e página de busca de serviços.

A página inicial disponibiliza links de navegação entre os módulos do sistema, enquanto a página de autenticação contém campos destinados ao envio de credenciais de acesso. Já a página de busca permite a realização de consultas por meio de parâmetros HTTP. Além das páginas principais, foram desenvolvidas páginas auxiliares sem funcionalidades reais, com o objetivo de aumentar a credibilidade do ambiente simulado e prolongar a interação de potenciais atacantes.

Todas as requisições recebidas, suas classificações e as respostas retornadas pela aplicação são registradas em arquivos de log no formato NDJSON. Esses registros podem futuramente ser integrados a plataformas SIEM para monitoramento e análise de eventos de segurança, permitindo acompanhar as interações realizadas no honeypot e as tentativas de exploração da aplicação.

3.3.1. *Deception* do Honeypot

O principal diferencial da ferramenta proposta é a incorporação de mecanismos de *deception*, responsáveis por gerar respostas dinâmicas de acordo com o comportamento identificado nas requisições. Essa estratégia aumenta o realismo do ambiente monitorado, favorecendo a observação de técnicas empregadas por atacantes humanos e a coleta de informações sobre ataques automatizados.

Inicialmente, cada requisição é submetida ao modelo de detecção. Caso seja classificada como maliciosa, ela é então encaminhada ao modelo de classificação do tipo de ataque, cujo resultado é utilizado pelo mecanismo de *deception* para selecionar a resposta apropriada.

As possíveis respostas estão descritas a seguir:

1. **Retorno Normal:** aplicada a requisições classificadas como benignas, retornando o comportamento esperado pela aplicação, como mensagens de autenticação inválida ou resultados de busca.
2. **Erro de SQL:** utilizado em requisições associadas a ataques SQL Injection, retornando mensagens simulando erros reais de sintaxe SQL, falhas de execução no banco de dados ou vazamentos fictícios de informações.
3. **Erro de XSS:** empregado em ataques classificados como Cross-Site Scripting, simulando comportamentos típicos de vulnerabilidades XSS refletidas e persistentes.
4. **Erro de RCE:** aplicado a ataques de Remote Code Execution, simulando a saída de comandos como *whoami* e *ls*.
5. **Erro de LFI:** exposto em ataques de Local File Inclusion, retornando conteúdos fictícios de arquivos sensíveis, como */etc/passwd* e */etc/shadow*.

O WebTrap também possui um módulo denominado *tracker*, responsável por acompanhar o histórico de interações de cada endereço IP e atribuir-lhe um nível de ameaça. À medida que esse nível aumenta, o sistema disponibiliza respostas que simulam vulnerabilidades progressivamente mais críticas, tornando a interação adaptativa. O *tracker* também registra essas informações em arquivos de log para posterior análise.

4. Resultados

Nesta seção são apresentados os resultados obtidos pelos dois modelos propostos: o modelo de detecção de tráfego malicioso e o modelo de classificação do tipo de ataque, ambos baseados em Random Forest. Esse algoritmo foi escolhido por ser menos suscetível a *overfitting* e apresentar altas taxas de precisão na identificação de ataques web [Ali and Karam 2026].

4.1. Avaliação dos Modelos de Aprendizado de Máquina

O modelo de detecção foi treinado utilizando o dataset completo, composto por 39.525 requisições HTTP. Para treinamento e validação, empregou-se uma divisão de 80% para treino e 20% para teste. Após sucessivos experimentos envolvendo ajuste de hiperparâmetros, refinamento do dataset, seleção de features e comparação entre algoritmos, obteve-se a configuração final. A Tabela 1 compara o desempenho das diferentes configurações avaliadas.

Tabela 1. Comparação entre configurações do modelo detector.

Algoritmo	Features	Acurácia	FP*	FN*
Random Forest	Somente TF-IDF	93,62%	2,1%	20,45%
Random Forest	Somente Numéricas	98,06%	37,75%	4,54%
Random Forest	**Combinação**	**97,90%**	**7,05%**	**6,81%**
XGBoost	Combinação	99,41%	29,4%	2,27%

* Taxas obtidas em testes externos independentes do conjunto de teste utilizado para calcular a acurácia.

Observa-se que a utilização isolada de features textuais ou numéricas resulta em desempenho inferior sob diferentes métricas, evidenciando que ambas capturam informações complementares durante o processo de classificação.

Embora o modelo baseado exclusivamente em atributos numéricos e o modelo XGBoost tenham apresentado acurácia ligeiramente superior, ambos produziram taxas significativamente maiores de falsos positivos. Em aplicações de segurança, esse comportamento compromete a confiabilidade do sistema, uma vez que tráfego legítimo passa a ser frequentemente classificado como malicioso. Por esse motivo, a combinação de features numéricas e textuais utilizando Random Forest foi adotada na versão final da ferramenta, por proporcionar melhor equilíbrio entre acurácia, falsos positivos e negativos.

A configuração final selecionada para a ferramenta alcançou acurácia geral de 97,90%, com precisão de 96,28% e 99,99%, recall de 99,99% e 95,39%, e F1-score de 98,10% e 97,64% para as classes benigna e maliciosa, respectivamente. Esses resultados demonstram elevada capacidade de distinguir tráfego legítimo de ataques, mantendo baixas taxas de falsos positivos e falsos negativos.

Além disso, a Tabela 2 apresenta as features de maior importância durante o treinamento, destacando atributos estruturais e padrões textuais associados a técnicas de evasão e ofuscação.

A análise das importâncias obtidas demonstra que tanto as features numéricas quanto as textuais desempenharam papel relevante na classificação das requisições. Observou-se que diversas features textuais mais relevantes estavam associadas a padrões

Tabela 2. 11 Principais features do modelo de detecção

Feature	Tipo	Importância
num_special_chars	numérica (F5)	7,02%
num_encoded	numérica (F6)	4,56%
req_length	numérica (F1)	4,09%
num_quotes	numérica (F4)	3,90%
+or+	TF-IDF	3,22%
+or	TF-IDF	2,78%
num_shell_ops	numérica (F11)	2,68%
num_shell_cmds	numérica (F13)	2,30%
%25	TF-IDF	2,11%
num_plus	numérica (F3)	1,85%
num_sqlwords	numérica (F8)	1,50%

frequentemente presentes em payloads maliciosos, como "+or+" e "%25". Esse comportamento evidencia a capacidade do modelo em identificar padrões relacionados a técnicas de evasão, codificação e ofuscação frequentemente utilizadas em ataques web.

O modelo de classificação do tipo de ataque foi treinado utilizando apenas as requisições maliciosas do dataset e empregando o mesmo conjunto de features e hiperparâmetros adotados pelo modelo de detecção, obtendo uma acurácia geral de 91,06%. Os resultados obtidos são apresentados na Tabela 3, evidenciando elevada capacidade de distinção entre as categorias SQL Injection e Cross-Site Scripting, possuindo métricas um pouco inferiores para Remote Code Execution e Local File Inclusion, devido a semelhança de características desses ataques.

Tabela 3. Métricas do modelo de classificação de ataques

Categoria	Precisão	Recall	F1-Score
SQLi	99,83%	95,56%	97,65%
XSS	98,09%	97,48%	97,78%
RCE	79,27%	82,94%	81,07%
LFI	82,43%	85,02%	83,70%

4.2. Testes com Ferramentas Automatizadas

Inicialmente, o modelo de detecção foi integrado ao honeypot web e avaliado com 16000 requisições benignas geradas automaticamente em Python utilizando a biblioteca *Faker*. O modelo apresentou taxa de classificação correta de 92,95%, sendo os falsos positivos concentrados em requisições contendo longas sequências numéricas e caracteres codificados.

Em seguida, foram realizados testes utilizando ferramentas amplamente empregadas em segurança ofensiva: *sqlmap* (SQL Injection), *XSSER* (Cross-Site Scripting), *Commix* (Remote Code Execution) e *Wfuzz* (Local File Inclusion) [Damele and Stampar , EPSYLON , Stasinopoulos et al. 2019, xmendez]. Cada ferramenta foi executada em diferentes configurações para aumentar a diversidade e complexidade dos ataques gerados.

O *sqlmap* foi executado contra as páginas de busca (GET) e autenticação (POST), empregando diferentes valores para os parâmetros *risk* e *level*, com o objetivo de gerar payloads mais sofisticados do que aqueles utilizados na construção do dataset. As cinco configurações empregadas (Tabela 4) produziram 5.018 requisições, das quais 99,41% foram corretamente classificadas.

Tabela 4. Comandos do *sqlmap* e quantidade de requisições geradas

Comando	Qtd. Requisições
`sqlmap -u http://192.168.1.2:5000/search?q= --level 5`	1296
`sqlmap -u http://192.168.1.2:5000/search?q= --risk 3`	165
`sqlmap -u http://192.168.1.2:5000/search?q= --level 5 --risk 3`	1592
`sqlmap -u http://192.168.1.2:5000/login? username= --level 5`	642
`sqlmap -u http://192.168.1.2:5000/login? username= --level 5 --risk 3`	1323

O *XSSER* foi avaliado em duas configurações (Tabela 5), gerando 2.794 requisições, das quais 99,67% foram corretamente identificadas.

Tabela 5. Comandos do *XSSER* e quantidade de requisições geradas

Comando	Qtd. Requisições
`xsser -u http://192.168.1.2:5000 -g 'search?q= XSS' --auto`	1294
`xsser -u http://192.168.1.2:5000/login -p 'username=XSS&pass=' --auto`	1500

O *Commix* também foi avaliado em duas configurações (Tabela 6), que geraram 3400 requisições, sendo 97,29% identificadas corretamente.

Tabela 6. Comandos do *Commix* e quantidade de requisições geradas

Comando	**Qtd. Requisições**
`commix -u http://192.168.1.2:5000/search?q=1 --level 3`	2300
`commix --method='POST' --data="username=&pass=" -u http://192.168.1.2:5000/login`	1100

Por fim, o *Wfuzz* foi utilizado em dois testes com wordlists próprias da ferramenta (Tabela 7), gerando 923 requisições, das quais 94,02% foram classificadas como ataque.

De forma geral, os resultados demonstram que o modelo manteve elevadas taxas de detecção mesmo diante de payloads produzidos por ferramentas amplamente utilizadas em testes de invasão. Além disso, a elevada precisão obtida indica que o conjunto de features desenvolvido foi capaz de representar padrões associados a diferentes técnicas de evasão e ofuscação.

Tabela 7. Comandos do *Wfuzz* e quantidade de requisições geradas

Comando	Qtd. Requisições
`wfuzz -c -z file,/usr/share/wordlists /wfuzz/vulns/dirTraversal.txt "http ://192.168.1.2:5000/search?q=FUZZ"`	847
`wfuzz -c -z file,/usr/share/wordlists /wfuzz/Injections/Traversal.txt "http ://192.168.1.2:5000/search?q=FUZZ"`	76

4.3. Avaliação do Honeypot Web

Além do modelo de detecção, também foi avaliado o classificador do tipo de ataque, que apresentou elevada capacidade de distinção entre SQL Injection e Cross-Site Scripting, com desempenho inferior para Remote Code Execution e Local File Inclusion, em razão da maior similaridade estrutural entre esses payloads.

Também foi analisado o desempenho computacional da ferramenta. O tempo médio de extração de features e classificação foi de 10 ms e 77 ms para o modelo de detecção e de 9 ms e 70 ms para o classificador de ataques, respectivamente, indicando baixa latência para execução integrada ao honeypot.

Os experimentos também evidenciaram que as ferramentas automatizadas não interpretam corretamente as respostas geradas pelo mecanismo de *deception*, limitando-se a registrar o sucesso das requisições enviadas. Em contrapartida, essas respostas tornam-se relevantes para atacantes humanos, aos quais são apresentados retornos simulando vulnerabilidades e informações falsas, favorecendo o prolongamento da interação com o honeypot.

Em conjunto, os resultados obtidos indicam que o *WebTrap* é capaz de detectar ataques gerados por ferramentas amplamente utilizadas em segurança ofensiva, identificar sua categoria com elevada precisão e fornecer respostas adaptativas por meio de mecanismos de *deception*, reunindo funcionalidades de detecção, classificação e interação com atacantes em uma única plataforma.

5. Discussão

O modelo de detecção utilizado pelo *WebTrap* apresentou elevadas taxas de detecção de ataques web por meio da análise dos métodos HTTP e dos parâmetros presentes nas requisições. Entretanto, a abordagem proposta possui limitações relacionadas ao escopo das informações analisadas, uma vez que o modelo não considera outros atributos relevantes das requisições HTTP, como cabeçalhos *User-Agent*, *cookies*, origem das conexões ou padrões temporais de navegação.

Outro aspecto importante refere-se ao ambiente controlado no qual o modelo foi desenvolvido e avaliado. O dataset utilizado concentrou-se principalmente em requisições relacionadas a mecanismos de busca e autenticação, cenários nos quais o modelo apresentou resultados satisfatórios. Contudo, aplicações web reais apresentam elevada diversidade estrutural e comportamental, incluindo parâmetros dinâmicos, diferentes padrões de navegação e múltiplos contextos de interação que não foram integralmente representados no conjunto de treinamento desenvolvido.

Dessa forma, embora os resultados obtidos indiquem elevada capacidade de detecção nos cenários avaliados, o modelo pode apresentar limitações de generalização principalmente na classificação de tráfego benigno em ambientes reais mais complexos e heterogêneos.

6. Conclusão

Este trabalho apresentou o desenvolvimento do *WebTrap*, uma ferramenta composta por um honeypot web integrado a dois modelos de aprendizado de máquina baseados em Random Forest. O primeiro modelo é responsável pela detecção de requisições maliciosas, enquanto o segundo realiza a classificação dos ataques em SQL Injection, Cross-Site Scripting, Remote Code Execution e Local File Inclusion. Dessa forma, a ferramenta auxilia na análise do comportamento de atacantes e na identificação de ameaças direcionadas a aplicações web. O código-fonte e os artefatos desenvolvidos encontram-se disponíveis publicamente em [pedro-hfw].

Durante o desenvolvimento da ferramenta, foi construído um dataset próprio composto por 39.525 requisições HTTP, empregado no treinamento dos modelos de detecção e classificação. Além disso, foram desenvolvidas features numéricas, textuais e relacionadas ao método HTTP, bem como realizados ajustes de hiperparâmetros, visando aumentar a capacidade de generalização e a detecção de ataques contendo técnicas de evasão e ofuscação.

A avaliação experimental foi realizada utilizando ferramentas amplamente empregadas na área de cibersegurança, como *sqlmap*, *XSSER*, *Commix* e *Wfuzz*. Os resultados demonstraram elevadas taxas de detecção de tráfego malicioso, atingindo cerca de 99% para ataques gerados pelo *sqlmap* e *XSSER*, 97% para ataques gerados pelo *Commix* e 94% para o *Wfuzz*. Nos testes envolvendo tráfego benigno, o modelo apresentou uma taxa de acerto de aproximadamente 93%.

Os resultados obtidos indicam que a integração entre aprendizado de máquina e mecanismos de *deception* constitui uma abordagem promissora para honeypots web inteligentes. A combinação de modelos de detecção e classificação permitiu identificar ataques e adaptar dinamicamente as respostas do ambiente monitorado, favorecendo tanto a coleta de informações quanto a interação com potenciais atacantes.

Como continuidade deste trabalho, pretende-se incorporar novos atributos das requisições HTTP, como cabeçalhos *User-Agent*, cookies e informações de sessão, além de ampliar e diversificar o dataset utilizado no treinamento, visando reduzir falsos positivos e aumentar a capacidade de generalização dos modelos em cenários reais. Adicionalmente, pretende-se investigar mecanismos capazes de distinguir atacantes humanos de ferramentas automatizadas, permitindo adaptar dinamicamente as estratégias de deception ao perfil observado durante a interação com o honeypot.

Agradecimentos

Esse trabalho foi desenvolvido com recursos do CEFET/RJ, por meio da bolsa de PIBIC.

Referências

Ali, R. H. and Karam, Z. S. (2026). Enhancing performance of intrusion prevention systems through machine learning: A comparative study. *Baghdad Science Journal*.

Buczak, A. L. and Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176.

Canadian Institute for Cybersecurity. Intrusion detection evaluation dataset (cic-ids2017). Disponível em `https://www.unb.ca/cic/datasets/ids-2017.html`. Acessado em maio de 2026.

Correia, P. H. B. and Pedrini, H. (2020). Detecção de domínios maliciosos baseada em técnicas de aprendizado de máquina. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação). Universidade Estadual de Campinas.

Damele, B. and Stampar, M. sqlmap: Automatic sql injection and database takeover tool. `https://sqlmap.org/`.

danielmiessler. Seclists. Disponível em `https://github.com/danielmiessler/seclists`. Acessado em maio de 2026.

EPSYLON. Xsser: Cross site "scripter". `https://github.com/epsylon/xsser`.

Faisal Fadlalla, F. and Elshoush, H. T. (2023). Input validation vulnerabilities in web applications: Systematic review, classification, and analysis of the current state-of-the-art. *IEEE Access*, 11:40128–40161.

Farzaan, M. A. M., Ghanem, M. C., El-Hajjar, A., and Ratnayake, D. N. (2024). Ai-enabled system for efficient and effective cyber incident detection and response in cloud environments. *arXiv preprint arXiv:2404.05602*.

Guarizi, B., Mascarenhas, D., and Moraes, I. (2025). Phishing guardian: Detecção de sites de phishing com machine learning. In *Anais do XXV Simpósio Brasileiro de Cibersegurança*, pages 693–709, Porto Alegre, RS, Brasil. SBC.

IBM. X-force threat intelligence index 2026. Disponível em `https://www.ibm.com/reports/threat-intelligence`. Acessado em maio de 2026.

Kaya, M. O., Dagdogen, H. A., Ozdem, M., and Das, R. (2026). An effective new penetration test approach to detect web attacks on web applications. *Expert Systems with Applications*, 298:129623.

Komiya, R., Paik, I., and Hisada, M. (2011). Classification of malicious web code by machine learning. In *2011 3rd International Conference on Awareness Science and Technology (iCAST)*, pages 406–411.

López, P. B., Pérez, M. G., and Nespoli, P. (2024). Cyber deception: State of the art, trends and open challenges.

Matthew Sudol. Web-application-attack-datasets. Disponível em `https://github.com/msudol/Web-Application-Attack-Datasets`. Acessado em maio de 2026.

McNally, S. and Curran, K. (2024). Web application vulnerabilities.

Nimmagadda, A. and Mehr, S. Y. (2025). Ai-powered intrusion detection system with honeypot integration. *International Journal of Intelligent Information Systems*.

Noman, M., Iqbal, M., and Manzoor, E. D. A. (2020). A survey on detection and prevention of web vulnerabilities. *International Journal of Advanced Computer Science and Applications*, 11:521–540.

OWASP Foundation. Owasp top 10:2025. Disponível em `https://owasp.org/Top10/2025/`. Acessado em maio de 2026.

Palo Alto Networks. 2026 unit 42 global incident response report - palo alto networks. Disponível em `https://www.paloaltonetworks.com/resources/research/unit-42-incident-response-report`. Acessado em maio de 2026.

pedro-hfw. Webtrap. Disponível em `https://github.com/pedro-hfw/webtrap-sbseg26`. Acessado em julho de 2026.

Rahman, M. (2025). Cross-domain semi-supervised detection of zero-day web application attacks via multi-modal http request and payload features.

Rong, W., Zhang, B., and Lv, X. (2018). Malicious web request detection using character-level cnn.

Sayari, A. and Rekhis, S. (2025). Cyber deception across domains: A comprehensive survey of techniques, challenges, and perspectives. *International Journal of Advanced Computer Science and Applications*, 16(7).

Sen, R., Heim, G., and Zhu, Q. (2022). Artificial intelligence and machine learning in cybersecurity: Applications, challenges, and opportunities for mis academics. *Communications of the Association for Information Systems*, 51:pp–pp.

Sharif, M. H. U. (2022). Web attacks analysis and mitigation techniques. *International Journal of Engineering Research & Technology (IJERT)*.

Stasinopoulos, A., Ntantogian, C., and Xenakis, C. (2019). Commix: automating evaluation and exploitation of command injection vulnerabilities in web applications. *Int. J. Inf. Secur.*, 18(1):49–72.

swisskyrepo. Payloadsallthethings. Disponível em `https://github.com/swisskyrepo/PayloadsAllTheThings`. Acessado em maio de 2026.

Thang, N. M. (2020). Improving efficiency of web application firewall to detect code injection attacks with random forest method and analysis attributes http request. *Programming and Computer Software*, 46(5):351–361.

xmendez. Wfuzz - the web fuzzer". `https://github.com/xmendez/wfuzz`.

Zhang, F., Zhou, S., Qin, Z., and Liu, J. (2003). Honeypot: a supplemented active defense system for network security. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 231–235.

Zhang, J., Zulkernine, M., and Haque, A. (2008). Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5):649–659.