Detecção Inteligente de Injeção de SQL integrando Ambientes de Nuvem e Borda

Michael S. Souza[1], **Silvio E. S. B. Ribeiro**[1], **Ivo A. Pimenta**[1], **Yanne O. Almeida**[1], **Francisco J. Cardoso**[1], **Rafael L. Gomes**[1]

[1]Universidade Estadual do Ceará (UECE), Brasil

{michael.souza,silvio.eduardo,ivo.aguiar,yanne.oliveira,

fco.cardoso}@aluno.uece.br, rafa.lopes@uece.br

Resumo. *Nos últimos anos a quantidade de serviços de computação urbana cresceu exponencialmente. Contudo, estes ainda são vulneráveis a potenciais ameaças de Injeção de SQL. Para lidar com este problema, soluções de segurança precisam, para além da eficiência na detecção, satisfazer requisitos de tempo de resposta e escalabilidade. Dentro deste contexto, este artigo propõe uma solução de detecção de Injeção de SQL baseada na integração entre ambientes de Borda e Nuvem, aos quais se aplicam técnicas de Filtragem por Expressões Regulares (RegEx) e Machine Learning (ML). A filtragem por RegEx no ambiente de Borda atua como uma primeira camada de proteção contra entradas maliciosas, melhorando o tempo de resposta da solução. Em seguida, o resultado da filtragem inicial é analisado por um modelo de ML para detectar SQLi com maior eficiência. Os experimentos realizados, utilizando um conjunto de dados reais, sugerem que a solução proposta detecta as ameaças de forma eficiente enquanto atende aspectos de escalabilidade e tempo de resposta.*

Abstract. *In recent years, the number of urban computing services has grown exponentially. However, these are still vulnerable to potential SQL Injection (SQLi) threats. Security solutions to deal with SQLi need, in addition to detection efficiency, to satisfy response time and scalability requirements. Within this context, this article proposes an SQLi detection solution based on the integration between Edge and Cloud environments, which apply Regular Expression (RegEx) filtering and Machine Learning (ML) techniques. RegEx filtering in the Edge environment acts as a first layer of protection against SQLi inputs, improving the solution's response time. Then, the result of the initial filtering is analyzed by an ML model to detect SQLi more efficiently. The experiments carried out, using a real data set, suggest that the proposed solution detects SQLi threats efficiently while meeting aspects of scalability and response time.*

1. Introdução

Nos últimos tempos, a expansão das zonas urbanas tem gerado inúmeras dificuldades práticas e de gestão. A Computação Urbana emerge neste contexto oferecendo soluções inovadoras e ajudando a resolver tais problemas, oriundos de grandes cidades e áreas metropolitanas [Silva et al. 2022]. Esta abordagem funciona por meio da implantação de serviços que utilizam informações massivas coletadas em grandes centros, provenientes de várias fontes heterogêneas [Portela et al. 2024, Funabiki 2011]. Os dados

são obtidos através de vários métodos de coleta, sendo crucial a utilização de técnicas de processamento e armazenamento, dada a grande quantidade de dados envolvidos, os inúmeros dispositivos ativos e a extensa troca de informações [Oliveira et al. 2020, Gomes et al. 2020].

Normalmente, os serviços de Computação Urbana utilizam Bancos de Dados Relacionais (BD) para guardar os dados coletados, comunicando-se através de uma Linguagem de Consulta Estruturada (*Structured Query Language* - SQL), incluindo comandos para apagar, alterar e criar entradas na base de dados [Geldenhuys et al. 2021, Silveira et al. 2023]. Existem ainda comandos para manipulação e consulta, que permitem inserir, eliminar, atualizar ou pesquisar informação em entidades de uma base de dados [Silveira et al. 2023].

Junto a este cenário, surge a necessidade de prover segurança a tais serviços, uma vez que ameaças exploram potenciais pontos fracos nos processos de comunicação e de acesso aos dados [Costa et al. 2020]. Uma vulnerabilidade notável é o acesso inseguro à informação por parte de utilizadores e programas nocivos, que tentam ultrapassar os serviços de computação urbana através de ataques de Injeção de SQL (*SQL Injection* - SQLi) [Das et al. 2019]. Estes ataques aproveitam lacunas de segurança nas entradas (como campos de texto e consultas), induzindo o sistema a processar os pedidos e, consequentemente, expondo dados sensíveis de clientes e dispositivos autenticados [Parashar et al. 2021]. Atualmente, de acordo com a *Open Worldwide Application Security Project* (OWASP), o SQLi é a principal ameaça à segurança dos serviços e aplicações na Internet [Tang et al. 2020].

Comandos SQLi podem ser estruturados de várias maneiras, potencialmente levando a respostas que beneficiam os atacantes, concedendo acesso indevido a dados privados, ou alterando/removendo dados na base de dados [Parashar et al. 2021, Li et al. 2019]. Atualmente, existem variadas soluções de segurança, tanto na literatura como no setor industrial, destinadas a detectar SQLi em serviços online e/ou nas bases de dados que estes utilizam. No entanto, estas soluções frequentemente priorizam a eficiência (precisão) na identificação de ataques de SQLi, ignorando o efeito destas medidas de segurança no desempenho de serviços de computação urbana. Por exemplo, técnicas de Inteligência Artificial (IA), principalmente Aprendizado de Máquina (*Machine Learning* - ML), têm sido usadas para a detecção de SQLi [Xie et al. 2019, Li et al. 2019, Parashar et al. 2021, Tang et al. 2020]. Entretanto, estas têm um tempo de resposta considerado longo, levando milissegundos para a análise de uma única entrada.

O atraso na detecção de ameaças influencia diretamente o desempenho do serviço, visto que muitos destes possuem restrições significativas, tanto de hardware quanto de software, e devem abordar aspectos de escalabilidade devido à grande quantidade de dados e comunicação envolvidos [Musznicki et al. 2022, Gomes et al. 2020, Geldenhuys et al. 2021, Lv et al. 2020]. Por este motivo, são necessárias soluções de segurança que, para além de serem eficientes na detecção de SQLi e de potenciais ameaças, cumpram também os critérios de tempo de processamento e tempo de resposta. Assim, uma abordagem adequada é analisar não todas as entradas dos serviços com técnicas de ML, mas apenas as entradas consideradas inicialmente como possíveis ameaças.

Neste contexto, este artigo apresenta uma solução composta por Expressões Regulares (*Regular Expressions* - RegEx) e Aprendizado de Máquina, denominada Dupla Camada de Detecção de SQLi (2LD-SQLi). O RegEx serve como um passo inicial de filtragem das ameaças de SQLi, com o objetivo de satisfazer os requisitos de tempo de resposta e escalabilidade. Em seguida, as entradas consideradas ameaças pelo RegEx são analisadas em profundidade por um modelo de ML para detectar casos de SQLi.

Em geral, a proposta visa determinar se uma consulta específica possui uma estrutura que sugere uma tentativa de SQLi, ou seja, busca filtrar as potenciais ameaças para evitar a execução desnecessária de modelos de ML para detecção, uma vez que estes têm um tempo de resposta elevado e consomem muitos recursos computacionais. Além disso, o 2LD-SQLi ajusta dinamicamente a ordem dos RegEx aplicados no processo de filtragem, com o objetivo de o acelerar. Assim, a solução proposta foca no equilíbrio entre a eficiência da detecção de SQLi e o tempo de resposta da solução para trazer escalabilidade ao processo de cibersegurança como um todo.

Objetivamente, o 2LD-SQLi aplica algumas estratégias: (I) Uma API para servir como um serviço de detecção de SQLi na borda, apoiando estratégias de cibersegurança em serviços de Computação Urbana; (II) Partição da solução em ambos os ambientes *Edge* e Nuvem, com foco na execução rápida e escalável da detecção na borda, mais próximo dos serviços de computação urbana, enquanto o treinamento das técnicas de ML e o gerenciamento do conjunto de RegEx, (tarefas custosas e de alta complexidade computacional) são realizadas na *Cloud*, dispondo de maior quantidade de recursos e alta disponibilidade de acesso; e, (III) Adaptabilidade do processo de detecção através da atualização do conjunto de RegEx (a estrutura das RegEx e a ordem dos testes), bem como da técnica de ML utilizada em conjunto.

Os resultados dos experimentos, utilizando um conjunto de dados real, indicam que a solução proposta possui uma eficiência adequada na detecção de ameaças de SQLi, oferecendo um tempo de resposta que satisfaz as exigências dos serviços de Computação Urbana. Foram avaliados vários cenários, testando a quantidade de RegEx na base de dados, bem como as requisições do processo de detecção de SQLi. Além disso, os resultados são comparados frente a métodos de *Machine Learning* utilizados na literatura, com o objetivo de estabelecer uma relação com outras soluções.

Em nosso trabalho anterior [Souza et al. 2023], propusemos um conjunto de RegEx para ser usado como um analisador de dados de entrada para identificar possíveis ataques de SQLi. No entanto, esse trabalho inicial não se concentrou na detecção de SQLi com alta precisão usando técnicas modernas, como o ML. Desta forma, a atual proposta tem os seguintes aspectos inovadores: (A) Integração da abordagem utilizando RegEx com técnicas de ML para efetuar a detecção de SQLi; (B) Evolução dos RegExes no conjunto, com foco em novos padrões de SQLi identificados; e, (C) Avaliação aprofundada com novos experimentos, considerando dados reais de SQLi disponíveis na literatura, além de novas métricas de avaliação.

A solução proposta foi desenvolvida durante projeto de Pesquisa e Desenvolvimento entre a UECE e o LACNIC[1], que busca desenvolver soluções industriais para

[1]https://programafrida.net/en/archivos/project/data-protection-system-based-on-anonymization-techniques-and-in-accordance-with-privacy-laws

proteção de dados sensíveis na Internet. A detecção de SQLi é um dos aspectos considerados no processo de proteção, uma vez que compromete a privacidade dos usuários.

O restante deste trabalho está organizado da seguinte forma: A Seção 2 apresenta soluções existentes na detecção de SQLi em sistemas gerais e no contexto de computação urbana. A Seção 3 descreve a solução de detecção de SQLi proposta. A Seção 4 discute os resultados dos experimentos realizados, e a Seção 5 conclui o trabalho.

2. Trabalhos Relacionados

Esta seção descreve os principais trabalhos relacionados ao tema e recentemente publicados pela comunidade científica, no âmbito da detecção de SQLi, além de soluções de segurança que utilizam técnicas de IA e integração entre Borda e Nuvem.

Parashar et al. [Parashar et al. 2021] propuseram um método para detectar injeção de SQL em aplicações de tecnologia da informação. Atingem este objetivo através da utilização da sumarização de texto, que permite o processamento de dados independentemente do tamanho da entrada. Ao criar um modelo de Machine Learning supervisionado a partir da sumarização, os autores conseguem automatizar a classificação de SQLi. No entanto, a abordagem do autor não considera questões de escalabilidade, o que pode levar a tempos de resposta mais altos, além de reduzir a aplicabilidade da solução em cenários de computação urbana.

Xie et al. [Xie et al. 2019] desenvolveram um método para detectar ataques de injeção de SQL utilizando *Elastic-Pooling Convolutional Neural Networks* (EP-CNNs). Este método cria uma matriz bidirecional que capta todos os dados sem truncagem e pode detectar eficazmente ataques de injeção de SQL, através da identificação de correspondências irregulares nos dados. Da mesma forma, Tang et al. [Tang et al. 2020] descrevem um método de detecção de SQLi baseado em Redes Neurais que visa atingir taxas de precisão elevadas, cerca de 99%. Os autores utilizaram técnicas estatísticas sobre entradas benignas e entradas maliciosas de SQL, derivadas de registros de acesso a URL de Provedores de Internet. Com base nos resultados estatísticos, os autores identificaram oito tipos de recursos e treinaram um modelo *Multi-Layer Perceptron* (MLP).

Li et al. [Li et al. 2019] propõem um modelo de Floresta Profunda para detectar ataques complexos de injeção de SQL (SQLi). O modelo baseia-se no algoritmo AdaBoost e concatena o vetor bruto de características com a média dos resultados anteriores como entrada para cada camada. A taxa de erro é utilizada para atualizar os pesos em cada camada, sendo atribuídos aos recursos pesos variáveis com base na sua influência e nos resultados durante o processo de treinamento. Os modelos propostos são eficientes na detecção de SQLi, mas têm custos computacionais elevados, resultando em tempos de processamento mais longos do que o ideal para a sistemas limitados de computação urbana. Numa linha semelhante, Fadolalkarim et al. [Fadolalkarim et al. 2020] apresentam o AD-PROM, um sistema de detecção de anomalias concebido para proteger bases de dados relacionais contra agentes maliciosos. O AD-PROM rastreia as chamadas de sistema executadas no sistema computacional que hospeda a base de dados alvo. Com base neste rastreio, o AD-PROM analisa os fluxos de controle e de dados dos programas, construindo um Modelo Oculto de Markov (*Hidden Markov Model* - HMM) que detecta anomalias que indicam ocorrências de SQLi no sistema hospedeiro da base de dados.

Crespo-Martinez et al. [Crespo-Martínez et al. 2023] demonstram que é possível

detectar dados do fluxo de entrada em ataques de SQLi a partir de informações de protocolo, através de dois conjuntos de dados baseados em ataques de SQLi. Depois disso, os autores avaliaram várias técnicas de Machine Learning para detectar situações de SQLi, onde a Regressão Logística apresentou os melhores resultados para os dados de fluxo de rede obtidos.

A partir das soluções existentes, é possível notar que nenhuma destas tem como objetivo desenvolver um método para detectar SQLi, levando em conta questões de escalabilidade e tempo de resposta, que é o foco da proposta 2LD-SQLi (conjunto RegEx flexível e adaptável integrado com modelo de ML).

3. Proposta

Esta secção detalha a proposta 2LD-SQLi, onde a Seção 3.1 descreve a arquitetura concebida para a solução, discutindo os módulos definidos e o seu fluxo de comunicação. A Seção 3.2 apresenta o contexto de ameaças de SQLi, o conjunto de RegEx definido para detectção, além da estratégia de ordenação dinâmica aplicada. Finalmente, a Seção 3.3 apresenta as técnicas de ML aplicadas na proposta.

3.1. Arquitetura de 2LD-SQLi

A solução 2LD-SQLi é composta por vários módulos: API, *Filtering*, *Detection*, *Training*, RegEx DB e *Update*. Além disso, consideramos os seguintes tipos de comunicação: comunicação interna que ocorre entre os módulos dentro do mesmo ambiente e comunicação via Internet que é realizada remotamente através da API (usando um canal seguro). Uma visão geral do 2LD-SQLi e do contexto de implantação é ilustrada na Figura 1.

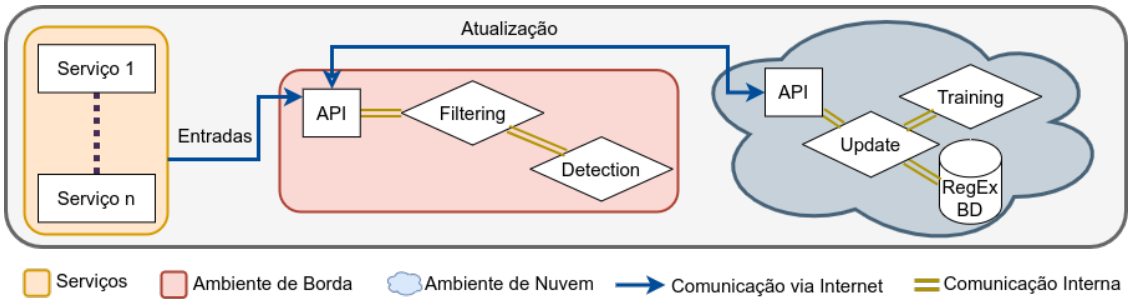

Figura 1. Visão geral do 2LD-SQLi.

A API permite o acesso à funcionalidade por serviços de Computação Urbana em uma abordagem interoperável, além de possibilitar a comunicação entre os ambientes de Borda e de Nuvem por meio de um canal seguro. No ambiente *Edge*, o módulo *Filtering* utiliza um conjunto de RegEx para realizar a filtragem inicial de potenciais ameaças de SQLi nos dados de entrada, enquanto o módulo *Detection* aplica um modelo ML previamente treinado no ambiente *Cloud*. Por outro lado, o módulo *Update* armazena e gere o conjunto completo de RegEx no RegEx DB, ainda no ambiente de Nuvem. Adicionalmente, o módulo *Update* executa o processo de atualização do modelo de ML utilizado no módulo *Detection* no ambiente de Borda, quando é acionado após o treinamento, que por sua vez é realizado pelo módulo *Trainning*. Adicionalmente, o módulo *Update* controla a

ordem das RegEx utilizadas no módulo *Filtering*, focando na ordem mais adequada para determinada situação. O fluxo dos passos executados pelo 2LD-SQLi está resumido na Figura 2.

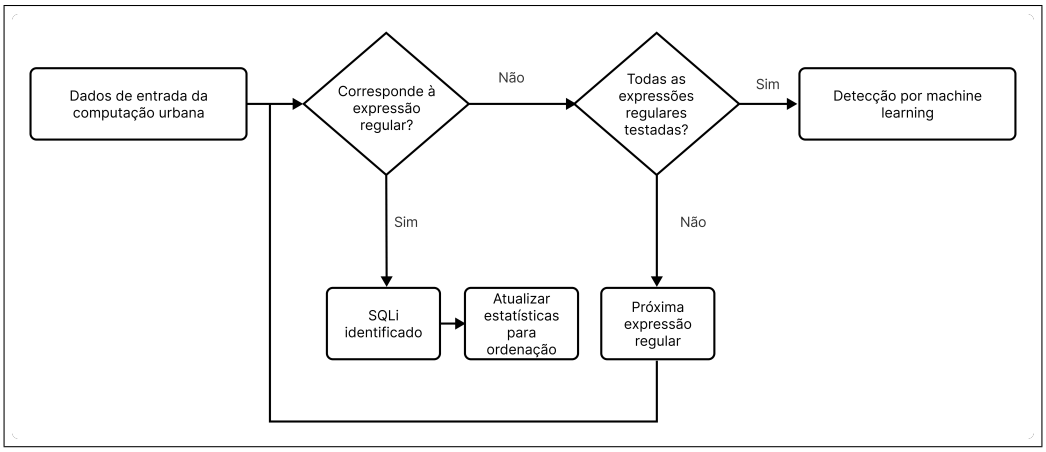

Figura 2. Etapas do 2LD-SQLi.

Considerando esta organização, a detecção de SQLi acontece no ambiente *Edge*, ou seja, ocorre mais próximo dos Serviços de Computação Urbana. O conjunto de RegEx utilizado pelo 2LD-SQLi é atualizado através da API com o ambiente de Nuvem (RegEx DB). Em seguida, descrevemos nas Seções 3.2 e 3.3, o conjunto Regex aplicado no módulo *Filtering* e os modelos ML utilizados no módulo *Detection*, respetivamente.

3.2. RegEx no 2LD-SQLi

Sistemas maliciosos de Injeção de SQL aproveitam de vulnerabilidades em sistemas de entrada de dados, empregando várias tácticas para enganar o sistema e expor informações de usuários autorizados e autenticados. É crucial compreender que o SQLi tem o potencial de utilizar indevidamente campos de nome de usuário e senha para realizar operações de linha de comando não autorizadas. Em particular, dispositivos de Internet das Coisas (IoT) podem funcionar como potenciais canais para ataques de injeção de SQL, sendo uma preocupação que se tornou mais acentuada com a expansão de serviços de computação dedicados. Cada variante de SQLi introduz um perfil de violação de segurança distinto, enfatizando a necessidade de compreender como as instruções SQL podem dar origem a estas vulnerabilidades. Sendo assim, esta etapa busca efetuar uma análise rápida e eficaz das possíveis ameaças de SQLi encontradas, empregando padrões de RegEx meticulosamente concebidos para abordar os perfis de ameaça que serviços de computação dedicados podem encontrar.

Existem vários tipos de ataques de SQLi conhecidos na literatura: Tautologias, Consultas Logicamente Incorretas, União, Baseado em Inferência, Codificação Alternada, etc [Lages and Pereira 2022, Yunus et al. 2018, Das et al. 2019]. Dessa forma, foram desenvolvidas RegEx para detectar padrões e características de cada tipo de SQLi. As RegEx definidas são:

1. Meta-caracteres: Pesquisa por meta-caracteres SQL específicos para identificar possíveis tentativas de injeção de SQL, incluindo caracteres hexadecimais como aspas simples, traço duplo e outros, que frequentemente marcam o início de um comentário. Regex: `(;%00)|(\%27)|(--[^\r\n]*)|(\')`

- `(;%00)`: Caractere ponto e vírgula seguido pelo caractere nulo.
- `(\%27)`: A string `%27`, que é a forma codificada por URL do caractere de aspas simples `(')`.
- `(--[\^\r\n]*)`: Qualquer caractere que começa com dois traços '−−', seguidos por quaisquer caracteres exceto quebras de linha.
- `(\')`: Caractere de aspas simples.

2. Meta-caractere alternativo: Busca por sinais de igual, aspas simples, traços duplos, ponto e vírgula e o caractere nulo.
   Regex: `((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)| (\%3B)|(;))`
   - `((\%3D)|(=))`: Caractere de sinal de igual ou sua versão codificada por URL `(%3D)`, que são usados em consultas SQL para denotar igualdade nas comparações.
   - `[\^\n]*`: Qualquer número de caracteres exceto o caractere de nova linha.
   - `((\%27)|(\'))`: Forma codificada por URL, ou não, do caractere de aspas simples.
   - `(--)`: Sequência de traço duplo usada para comentar consultas SQL.
   - `((\%3B)|(;))`: Caractere ponto e vírgula, ou sua versão codificada por URL, que separa múltiplas instruções SQL em uma única consulta.
3. "OR"dentro de outras strings: Pesquisa pela palavra 'or' e em formato codificado por URL.
   Regex: `((\%27)|(\'))((\%6F)|o|(\%4F))((\%72) |r|(\%52))`
   - `((\%27)|(\'))`: Aspas simples simples ou codificadas por URL.
   - `((\%6F)|o|(\%4F))`: Um 'o' codificado por URL, um 'o' simples ou um 'O' maiúsculo codificado por URL.
   - `((\%72)|r|(\%52))`: Um 'r' codificado por URL, um 'r' simples ou um 'R' maiúsculo codificado por URL.
4. Operadores lógicos: Procura pelas palavras 'and' ou 'or' seguidas por espaços.
   Regex: `(\W)(and|or)\s*`
   - `(\W)`: Caractere especial (não é um dígito, letra ou sublinhado).
   - `(and|or)`: 'and' ou 'or'.
   - `\s*`: Espaços em branco.
5. Instrução 'UNION': Procura pela palavra 'union' e em formato codificado por URL.
   Regex: `((\%27)|(\'))UNION`
   - `((\%27)|(\'))`: Codificação usual ou codificada por URL de uma aspas simples.
   - `UNION`: Palavra 'UNION'.
6. Consultas SQL: Procura por palavras-chave SQL.
   Regex: `([\s\(\)])*(create|select|delete|update |drop|alter|insert)([\s\(\)])*`
   - `([\s\(\)])*`: Qualquer ocorrência de caracteres de espaço em branco ou parênteses.
   - `(create|select|delete|update|drop|alter|insert)`: Uma das palavras-chave SQL 'create', 'select', 'delete', 'update', 'drop', 'alter' ou 'insert'.
7. 'exec' e 'execute': Procura pelas palavras-chave 'exec' ou 'execute'.
   Regex: `([\s\(\)])*(execute|exec)([\s\(\)])*`

- `([\s\(\)])*`: Qualquer ocorrência de qualquer caractere de espaço em branco ou parênteses.
- `(exec|execute)`: Palavras-chave 'exec' ou 'execute'.

8. Operador AND: Procura por AND em vários formatos, como símbolo de mais, operadores lógicos e formato codificado.
   Regex: `(\%20and|\+and|&&|\&\&)`
   - `(\%20and)`: Caractere de espaço codificado por URL.
   - `(+and)`: Caractere de espaço codificado por URL como parâmetro de consulta.
   - `(&&)` e `(\&\&)`: Operador lógico 'and' das linguagens de programação.

Estas RegExs aplicam a captura de múltiplos padrões, com o objetivo de identificar o maior número possível de ameaças de SQLi. A aplicação de várias expressões regulares para filtrar a entrada de dados melhora a segurança de uma aplicação, prevenindo e detectando potenciais ataques SQLi, uma vez que utilizam codificação, sintaxe ou combinações de texto distintas para ultrapassar as ferramentas de segurança.

Além disso, o 2LD-SQLi ajusta dinamicamente a ordem das RegEx testadas na entrada de dados, reordenando-as de acordo com as correspondências ocorridas, ou seja, o 2LD-SQLi traz para a frente as RegEx que estão atingindo mais ameaças de SQLi no processo de filtragem (em nossos experimentos, configuramos o processo de reordenação após cada análise de 100 cadeias). O ajuste dinâmico da ordem das RegEx traz vários benefícios para o processo de filtragem. Dentre estes, diferentes cenários podem exigir que diferentes padrões de RegEx sejam aplicados primeiro, uma vez que os dados de entrada podem mudar com o tempo. Portanto, o ajuste dinâmico da ordem dos testes de RegEx ajuda na adaptação a novos padrões e estruturas de dados, sem a necessidade de uma reconfiguração completa. Além disso, identifica rapidamente possíveis ameaças de SQLi, acelerando o processo de filtragem e reduzindo a carga nos testes de RegEx subsequentes, que consomem mais recursos.

3.3. Modelo de Machine Learning

O processo de treinamento do modelo de ML envolve a inserção de entradas de texto e, em seguida, a implementação da técnica de ML específica. Cada técnica de ML emprega uma abordagem distinta para interpretar os dados.

Visto que o mecanismo proposto funciona de forma independente da técnica de ML existente, as seguintes técnicas foram consideradas neste artigo: (I) Regressão Logística (*Logistic Regression* - LR) é utilizada em tarefas de classificação binária, como a detecção de SPAM ou o diagnóstico médico, estimando a probabilidade de um resultado binário, que depende de uma ou mais variáveis preditoras. (II) Floresta Aleatória (*Random Forest* - RF) é um conjunto coletivo de árvores de decisão. Esta abordagem encontra aplicação em tarefas de classificação e regressão, uma vez que reúne várias árvores de decisão para aumentar a precisão e atenuar o *overfitting*; (III) Árvore de Decisão (*Decision Tree* - DT) é um modelo em forma de árvore em que cada nó interno simboliza uma caraterística, cada ramo significa uma regra de decisão e cada folha representa um determinado resultado. (IV) Rede Neural Convolucional (*Convolutional Neural Network* - CNN), utiliza camadas convolucionais para aprender automaticamente características a partir dos dados, tornando-as poderosas para tarefas como a classificação de imagens e a detecção de anomalias; Por fim, (V) Máquina de Vetores de Suporte (*Support Vector*

Machine - SVM), que foi concebida para buscar um hiperplano ótimo para segregar eficazmente dados em classes distintas, sendo aplicável em espaços de grandes dimensões e para aplicações como classificação de imagens.

4. Experimentos Realizados

O repositório[2] dá acesso ao método proposto. Este método foi rigorosamente testado, como detalhado nesta seção, com intuito de determinar a sua capacidade de identificar ameaças de SQLi. Para os testes, foi criado um ambiente específico, enriquecido com um conjunto de dados que contém possíveis ameaças de SQLi. Este ambiente abrange várias configurações computacionais, incluindo máquinas virtuais hospedadas na nuvem e sistemas de hardware tradicionais. A implantação do 2LD-SQLi em cenários reais é o foco principal dos testes. Este arranjo oferece informações sobre o desempenho e as implicações do método. Enquanto a Subseção 4.1 fornece uma visão geral abrangente do design experimental, os resultados derivados são explorados na Subseção 4.2.

4.1. Configuração dos Experimentos

Os experimentos utilizaram ambientes de teste que incluíam recursos de Computação em Nuvem e máquinas físicas. É crucial testar uma solução em várias configurações de Nuvem para confirmar a sua resiliência, escalabilidade e confiabilidade [Costa et al. 2020]. Foi utilizada a Huawei Cloud[3] como o ambiente de Nuvem, que por sua vez oferece uma gama de máquinas virtuais *Elastic Cloud Server* (ECS)[4]. Notavelmente, o ECS na Huawei Cloud foi configurado com 16 GB de memória e 4 CPUs Virtuais (vCPUs). Nestas condições, consideramos as seguintes máquinas: *General Computing Plus* (GCP), nível básico de desempenho de vCPU; *Memory Optimized* (MO), alto desempenho de acesso à memória; *High-Performance Computing* (HPC), foco em computação paralela e serviços de infraestrutura de alto desempenho; Kunpeng (KPG), linha de base de desempenho de vCPU com processador Kupeng 920; e, *Physical Machine* (PM), máquina usual com processador CPU Intel Core i7-12700, com um disco SSD, GPU RTX 3060 e 16GB de RAM DDR5 4400MHz.

Para os experimentos foi utilizado um conjunto de dados de ameaças de SQLi[5] amplamente utilizado na literatura [Rahul et al. 2021, Devalla et al. 2022, Hosam et al. 2021, Roy et al. 2022]. Este conjunto de dados tem 19340 entradas, das quais 7962 são entradas benignas, enquanto 11378 são SQLi. Assim, é possível ter uma avaliação completa da proposta.

Foram analisadas diversas técnicas de ML[6], de acordo com a Seção 3.3, com intuito avaliar a capacidade de detecção do 2LD-SQLi. Também foi avaliada a eficiência de tempo da solução na detecção e atualização das RegEx, onde foram feitas 100 repetições com um intervalo de confiança de 95%. Além disso, foram consideradas as seguintes métricas para avaliar o desempenho da detecção: Acurácia (em percentual), taxa de classificações corretas; Recall (em percentual), eficiência em detectar corretamente a en-

[2]https://github.com/538Michael/

[3]huaweicloud.com

[4]support.huaweicloud.com/intl/en-us/productdesc-ecs/ecs_01_0073.html

[5]kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

[6]kaggle.com/code/chinonsocynthia/sql-inject-using-linear-models-and-cnn

trada analisada; Precisão (em percentual), capacidade de acertar quais dos valores positivos são realmente positivos; e F1-Score (em percentual), a média harmônica da Precisão e Recall.

4.2. Resultados

A seguir, discutimos os experimentos realizados. A Figura 3 apresenta o tempo de processamento necessário para analisar cadeias de entrada. A Figura 4 mostra a detecção das técnicas de ML independentes frente a etapa RegEx do 2LD-SQLi. A Figura 5 ilustra o desempenho de toda a execução do 2LD-SQLi ao adicionar as técnicas de ML. Por fim, as Tabelas 1 e 2 apresentam as métricas de avaliação da Acurácia, Precisão, F1-Score e Recall para comparativo da eficiência do 2LD-SQLi em relação às técnicas de ML existentes aplicadas de forma independente, bem como a melhoria das mesmas técnicas quando incorporadas na solução.

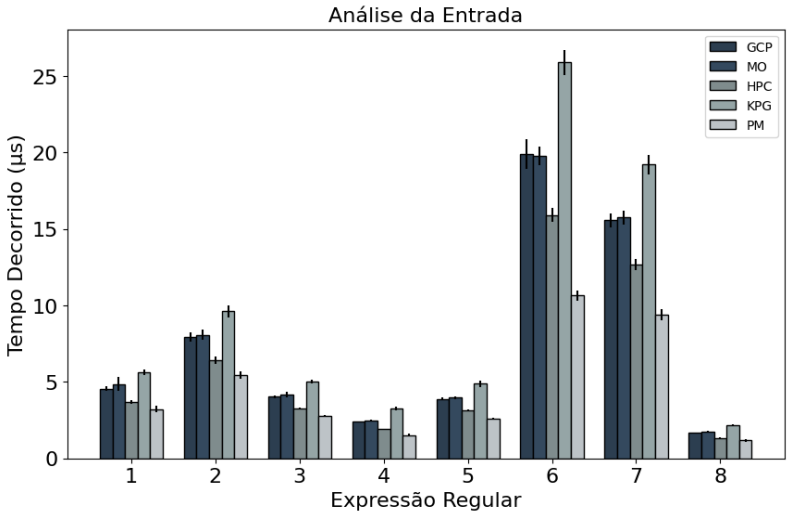

Figura 3. Tempo de cada RegEx para analisar os dados de entrada.

Conforme mostrado na Figura 3, o tempo necessário para a análise de entradas é consistente em diferentes ambientes, abrangendo máquinas virtuais e físicas. No entanto, são observados dois pontos-chave: (i) A máquina física (PM) apresenta melhor desempenho devido a alta capacidade do hardware e o seu processamento na borda, um cenário que não se aplica às máquinas virtuais na nuvem (GCP, MO, HPC e KPG) que possuem maior atraso nas requisições e atualizações. (ii) Conforme descrito na secção 3.2, os RegEx 6 e 7 demoram efetivamente mais tempo de processamento devido às suas estruturas mais complexas. Ainda assim, a solução tem um tempo de análise adequado que satisfaz os requisitos de escalabilidade ao analisar 1000 requisições em cerca de 10 milissegundos, um tempo significativamente inferior ao da comunicação de dados na Internet [M and H B 2022].

Além disso, o tempo de filtragem do 2LD-SQLi frente ao tempo de detecção das técnicas de ML são ilustrados na Figura 4. O tempo decorrido de filtragem utilizando o mesmo conjunto de RegEx da solução é cerca de 23% mais rápido do que os modelos que apresentaram melhor desempenho (LR e DT). Este fato representa a capacidade da solução de acelerar o processo de detecção, atualizando o conjunto de RegEx, deixando os casos não capturados nesta etapa para os modelos de ML na etapa seguinte.

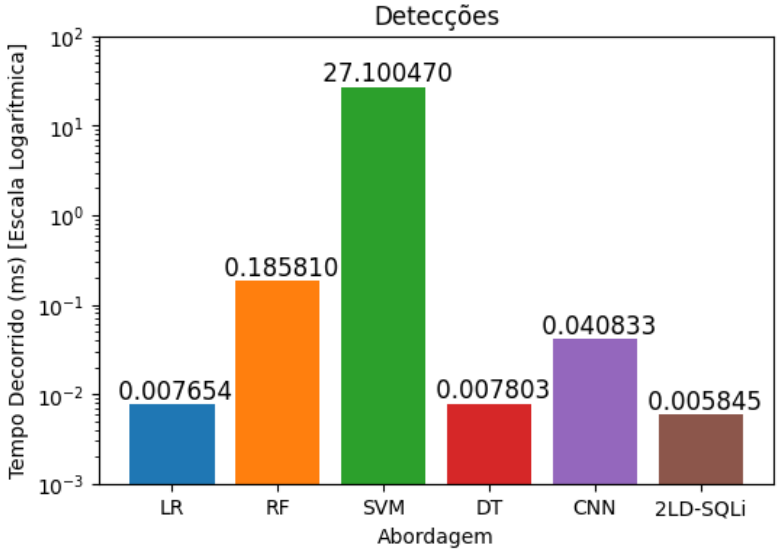

Figura 4. Tempo de filtragem do 2LD-SQLi e tempo de detecção das técnicas de ML.

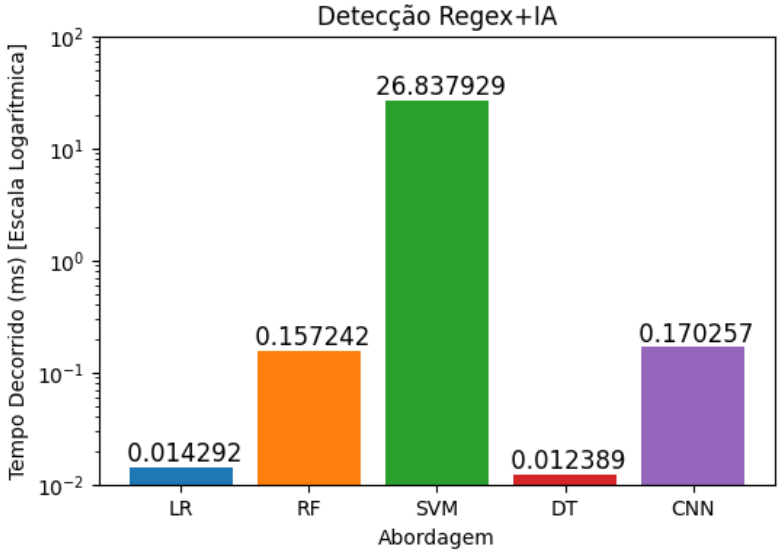

Figura 5. Tempo de Detecção do 2LD-SQLi para Filtragem e Análise

No último experimento, foi analisado o tempo decorrido de todo o processo do 2LD-SQLi para detectar SQLi, ou seja, o tempo decorrido para testar todo o conjunto de RegEx no *Filtering* e, quando nenhum dos RegEx resulta em correspondência, a execução do modelo ML para realizar a decisão final. Desta forma, os resultados da Figura 5 representam os casos de ambas as etapas (RegEx e ML) quando o conjunto de RegEx não é capaz de identificar um SQLi (verdadeiro positivo) ou a entrada de dados é um caso verdadeiro negativo de SQLi. É possível notar que, nestas situações, a abordagem mais adequada é aplicar os modelos DT ou LR, uma vez que estes têm um tempo de processamento inferior. Além disso, um aspecto relevante é o fato do tempo de detecção do modelo CNN para um caso negativo verdadeiro ser inferior ao tempo médio de detecção quando todos os casos são incluídos (como mostra a Figura 4).

Tabela 1. Desempenho da Detecção

Abordagem	F1-Score	Precisão	Revocação	Acurácia
LR	0.94	1.00	0.89	0.93
RF	0.94	0.99	0.89	0.93
SVM	0.81	1.00	0.68	0.82
DT	0.94	0.99	0.90	0.94
CNN	0.95	0.99	0.92	0.95
2LD-SQLi	0.98	0.99	0.97	0.98

Tabela 2. Desempenho de Detecção do 2LD-SQLi (RegEx+ML)

Abordagem	F1_Score	Precisão	Revocação	Acurácia
RegEx+LR	0.99	1.00	0.98	0.99
RegEx+RF	0.99	1.00	0.99	0.99
RegEx+SVM	0.98	1.00	0.97	0.98
RegEx+DT	0.99	1.00	0.99	0.99
RegEx+CNN	0.99	1.00	0.99	0.99

Os dados apresentados na Tabela 1 ilustram o sucesso das soluções avaliadas em identificar se uma entrada constitui ou não uma ameaça de SQL Injection. No caso do 2LD-SQLi, estes resultados realçam a sua capacidade de reconhecer potenciais ameaças e o valor da adoção de soluções mais complexas para poupar tempo e recursos computacionais. Finalmente, na Tabela 2, analisamos os benefícios da integração da abordagem RegEx e dos modelos de ML. De acordo com a Figura 5, no que diz respeito ao processamento, a abordagem mais adequada é utilizar modelos DT ou LR no módulo *Detection* do 2LD-SQLi. Da mesma forma, quando se analisam os dados apresentados na Tabela 2, é possível concluir que a opção mais adequada é utilizar o modelo DT na solução, que atinge os valores mais elevados em todas as métricas avaliadas.

A partir desses resultados, é possível notar que a eficiência do 2LD-SQLi, tanto em relação ao tempo de resposta quanto à eficiência da detecção, supera a abordagem existente de aplicação de ML independente para detectar SQLi. Por essa razão, a utilização do 2LD-SQLi como ferramenta de cibersegurança para a primeira linha de defesa em ambientes *Edge* pode ser crucial para apoiar os serviços de computação urbana, uma vez que apresenta um tempo de resposta reduzido e uma elevada precisão no processo de detecção.

5. Conclusão

Os serviços de computação urbana tornaram-se uma realidade nos últimos anos devido à expansão das tecnologias de comunicação e informação nas cidades e centros metropolitanos. Esses serviços geram um volume massivo de dados armazenados em BDs SQL. Esse cenário aumentou a possibilidade de ameaças, principalmente SQLi. Para fazer frente a essa realidade, este artigo apresenta o 2LD-SQLi, uma solução de detecção de ameaças SQLi baseada em RegEx que tem como objetivo atuar como um serviço de filtragem inicial para proteção contra ameaças SQLi para atender aos requisitos de tempo de resposta e escalabilidade. Posteriormente, as entradas consideradas como ameaças são analisadas por um modelo de *Machine Learning*. Como trabalho futuro, pretendemos desenvolver um novo RegEx para ser adicionado ao conjunto do processo de filtragem.

Adicionalmente, iremos realizar um *Data Mining* no conjunto de dados SQLi, aplicando algoritmos de clusterização para compreender melhor o perfil das entradas, bem como a capacidade da solução para detectar cada grupo.

Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) do Brasil (N^o 303877/2021-9) e Centro de Informação da Rede da América Latina e Caribe (LACNIC) através do *FRIDA Grant* pelo apoio financeiro.

Referências

Costa, W. L., Silveira, M. M., de Araujo, T., and Gomes, R. L. (2020). Improving ddos detection in iot networks through analysis of network traffic characteristics. In *2020 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6.

Crespo-Martínez, I. S., Campazas-Vega, A., Guerrero-Higueras, Á. M., Riego-DelCastillo, V., Álvarez-Aparicio, C., and Fernández-Llamas, C. (2023). Sql injection attack detection in network flow data. *Computers & Security*, 127:103093.

Das, D., Sharma, U., and Bhattacharyya, D. K. (2019). Defeating sql injection attack in authentication security: an experimental study. *International Journal of Information Security*, 18(1):1–22.

Devalla, V., Srinivasa Raghavan, S., Maste, S., Kotian, J. D., and Annapurna, D. D. (2022). murli: A tool for detection of malicious urls and injection attacks. *Procedia Computer Science*, 215:662–676. 4th International Conference on Innovative Data Communication Technology and Application.

Fadolalkarim, D., Bertino, E., and Sallam, A. (2020). An anomaly detection system for the protection of relational database systems against data leakage by application programs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 265–276.

Funabiki, N. (2011). *Wireless Mesh Networks*. IntechOpen.

Geldenhuys, M. K., Will, J., Pfister, B. J. J., Haug, M., Scharmann, A., and Thamsen, L. (2021). Dependable iot data stream processing for monitoring and control of urban infrastructures. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 244–250.

Gomes, R. L., Bittencourt, L. F., and Madeira, E. R. M. (2020). Reliability-aware network slicing in elastic demand scenarios. *IEEE Communications Magazine*, 58(10):29–34.

Hosam, E., Hosny, H., Ashraf, W., and Kaseb, A. S. (2021). Sql injection detection using machine learning techniques. In *2021 8th International Conference on Soft Computing Machine Intelligence (ISCMI)*, pages 15–20.

Lages, G. and Pereira, R. (2022). Estudo comparativo entre tecnicas de detecccao e prevencao de ataques de injecao sql. In *Anais do XVII Escola Regional de Banco de Dados*.

Li, Q., Li, W., Wang, J., and Cheng, M. (2019). A sql injection detection method based on adaptive deep forest. *IEEE Access*, 7:145385–145394.

Lv, Z., Hu, B., and Lv, H. (2020). Infrastructure monitoring and operation for smart cities based on iot system. *IEEE Transactions on Industrial Informatics*, 16(3):1957–1962.

M, G. and H B, P. (2022). Semantic query-featured ensemble learning model for sql-injection attack detection in iot-ecosystems. *IEEE Transactions on Reliability*, 71(2):1057–1074.

Musznicki, B., Piechowiak, M., and Zwierzykowski, P. (2022). Modeling real-life urban sensor networks based on open data. *Sensors*, 22(23).

Oliveira, D. H. L., Filho, F. M. V., de Araújo, T. P., Celestino, J., and Gomes, R. L. (2020). Adaptive model for network resources prediction in modern internet service providers. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6.

Parashar, D., Sanagavarapu, L. M., and Reddy, Y. R. (2021). Sql injection vulnerability identification from text. In *14th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ISEC 2021, New York, NY, USA. Association for Computing Machinery.

Portela, A. L. C., Ribeiro, S. E. S. B., Menezes, R. A., de Araujo, T., and Gomes, R. L. (2024). T-for: An adaptable forecasting model for throughput performance. *IEEE Transactions on Network and Service Management*.

Rahul, S., Vajrala, C., and Thangaraju, B. (2021). A novel method of honeypot inclusive waf to protect from sql injection and xss. In *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, volume 1, pages 135–140.

Roy, P., Kumar, R., and Rani, P. (2022). Sql injection attack detection by machine learning classifier. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 394–400.

Silva, M. V., Mosca, E. E., and Gomes, R. L. (2022). Green industrial internet of things through data compression. *International Journal of Embedded Systems*, 15(6):457–466.

Silveira, M. M., Portela, A. L., Menezes, R. A., Souza, M. S., Silva, D. S., Mesquita, M. C., and Gomes, R. L. (2023). Data protection based on searchable encryption and anonymization techniques. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5.

Souza, M., Ribeiro, S., and Gomes, R. (2023). Detecção de ameaças de injeção de sql em serviços de computação urbana. In *Anais do VII Workshop de Computação Urbana*, pages 145–158, Porto Alegre, RS, Brasil. SBC.

Tang, P., Qiu, W., Huang, Z., Lian, H., and Liu, G. (2020). Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528.

Xie, X., Ren, C., Fu, Y., Xu, J., and Guo, J. (2019). Sql injection detection for web applications based on elastic-pooling cnn. *IEEE Access*, 7:151475–151481.

Yunus, M. A. M., Brohan, M. Z., Nawi, N. M., Surin, E. S. M., Najib, N. A. M., and Liang, C. W. (2018). Review of sql injection: Problems and prevention. *JOIV: International Journal on Informatics Visualization*, 2(3-2):215–219.