Detecção de Ameaças de Injeção de SQL em Serviços de Computação Urbana

Michael S. Souza[1], **Silvio E. Ribeiro** [1], **Rafael L. Gomes**[1]

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

{michael.silva,silvio.eduardo}@aluno.uece.br, rafa.lopes@uece.br

Resumo. *Devido ao expressivo volume de dados heterogêneos gerados pelos espaços urbanos, bem como aos avanços da tecnologia da informação, diversos Serviços de Computação Urbana têm ganhado visibilidade. Estes serviços utilizam bancos de dados relacionais para armazenamento dos dados coletados e são suscetíveis a possíveis ameaças, dentre elas os ataques de SQL Injection (SQLi). Desta forma, se fazem necessárias soluções de segurança que possam, além de trazer eficiência na detecção, atender aos aspectos de tempo de processamento e tempo de resposta. Dentro deste contexto, este artigo apresenta uma solução para Detecção Escalável de Ameaças SQLi (DEA-SQLi) baseada em Expressões Regulares, atuando assim como um serviço de filtragem inicial para proteção contra ameaças de SQLi, a fim de atender a demandas de tempo de resposta e escalabilidade. Os resultados dos experimentos utilizando um conjunto de dados real sugerem que o DEA-SQLi possui uma eficiência adequada para a detecção de ameaças, enquanto atende aos requisitos de escalabilidade dos serviços de computação urbana.*

Abstract. *Due to the significant volume of heterogeneous data generated by urban spaces, as well as advances in information technology, several Urban Computing Services have gained visibility. These services use relational databases to store the collected data, where these databases are susceptible to possible threats, including SQL Injection (SQLi) attacks. Therefore, security solutions are needed that can, in addition to bringing efficiency in detection, meet the processing time aspects of this detection. Within this context, this article presents a solution for SQLi Scalable Threat Detection (DEA-SQLi) based on Regular Expressions, acting as an initial filtering service for protection against SQLi threats in order to address aspects of response time and scalability. Results of experiments using a real dataset suggest that DEA-SQLi has adequate SQLi threat detection efficiency, while meeting the scalability aspects of urban computing services.*

1. Introdução

Nos últimos anos, o rápido crescimento das cidades e espaços urbanos vêm resultando em muitos desafios técnico-operacionais e de administração. A Computação Urbana surgiu a fim de propor soluções modernas e contribuir para a superação desses desafios enfrentados pelas grandes cidades e regiões metropolitanas. Este paradigma atua através da implantação de serviços que utilizam informações oriundas de um volume massivo de dados heterogêneos coletados em grandes cidades, e de diversas fontes [Rodrigues et al. 2019]. A coleta de dados ocorre através de diferentes

tecnologias de sensoriamento remoto, técnicas de gerenciamento e modelos analíticos [Musznicki et al. 2022]. Uma vez que esses dados são coletados, deve-se utilizar técnicas de processamento e armazenamento, já que isto envolve a manipulação de um grande volume de dados oriundos de grandes centros urbanos, com muitos dispositivos operantes, e bastante troca de informação.

Em geral, serviços de computação urbana possuem Sistemas de Gerenciamento de Banco de Dados (SGBD) que operam sobre bancos de dados relacionais para armazenamento dos dados coletados. Em contextos urbanos, diversas ferramentas ou serviços de monitoramento e troca de informações entre dispositivos são usadas para este fim. [Geldenhuys et al. 2021, Lv et al. 2020]. A forma mais usual de uma aplicação interagir com um banco de dados é através de uma Linguagem Estruturada de Consultas (SQL - *Structured Query Lenguage*) que possui, entre outras categorias, comandos para definição de dados, que são responsáveis pela criação, deleção, alteração, etc, em entidades do banco de dados. Assim como, também existem comandos de manipulação e consulta sobre os dados, que inserem, deletam, atualizam ou buscam por informações em tabelas do banco, tornando possível o armazenamento e acesso facilitado aos diversos tipos de dados presentes em sistemas de computação urbana e, consequentemente, viabilizando a comunicação entre diversos dispositivos de borda, *hosts* e servidores.

Neste cenário de larga utilização de serviços de computação, insurgem demandas de segurança que tornam os dispositivos e os usuários alvos de diversos tipos de ameaças e sistemas maliciosos, que visam explorar possíveis vulnerabilidades no processo de comunicação e de acesso aos dados. Entre estas vulnerabilidades, destacam-se o acesso inseguro às informações, por usuários e programas mal intencionados, que visam burlar os serviços de comunicação urbana através de ataques de Injeção de SQL (*SQL Injection* - SQLi) [Kumar and Pateriya 2012], que exploram brechas de segurança em consultas e campos de texto, enganando o sistema que processa as requisições, através de técnicas pervasivas que expõem dados, muitas vezes sensíveis, de clientes e dispositivos autenticados [Parashar et al. 2021].

Uma característica usual dos ataques de SQLi é a utilização de campos de usuário e senha como meio para a execução de comandos não autorizados, entretanto, não limitado a este cenário, pois usuários, dispositivos e sistemas autenticados podem ser infectados e se tornarem atacantes, i.e., sistemas que vão tentar explorar vulnerabilidades através de SQLi [M and H B 2022]. Assim, esses comandos possuem diversas possibilidades de estruturação e, consequentemente, podem provocar respostas favoráveis aos atacantes, fornecendo acesso indevido a dados confidenciais, alterar ou apagar dados existentes no banco de dados, dentre outros prejuízos. Dentro deste cenário, tem-se que os dispositivos IoT tem sido usados como vetor de execução de SQLi, principalmente com a expansão dos serviços de computação Urbana [Rizvi et al. 2018].

Atualmente, existem diversas soluções de segurança na literatura e no meio industrial que visam detectar SQLi em serviços online e/ou no acesso aos bancos de dados que estes serviços utilizam. Contudo, estas soluções tendem a focar na eficiência (acurácia) em detectar os ataques SQLi, desconsiderando o impacto destas soluções de segurança no desempenho dos serviços de computação urbana. Por exemplo, técnicas custosas de Inteligência Artificial têm sido aplicadas para detecção de SQLi [Xie et al. 2019, Li et al. 2019, Parashar et al. 2021, Tang et al. 2020], mas estas possuem um tempo de

resposta da detecção considerado alto (alcançando cerca de milissegundos para realizar uma única análise). Este impacto no tempo de resposta afeta diretamente o desempenho dos serviços, visto que muitos destes possuem muitas restrições, e necessitam atender aos aspectos de escalabilidade, devido ao grande volume de dados e a comunicação realizada [Musznicki et al. 2022, Geldenhuys et al. 2021, Lv et al. 2020]. Sendo assim, faz-se necessário soluções de segurança que possam, além de trazer eficiência na detecção de SQLi e possíveis ameaças, atender aos aspectos de tempo de processamento e tempo de resposta.

Dentro deste contexto, este artigo apresenta uma solução para Detecção Escalável de Ameaças de SQLi (DEA-SQLi) baseada em (*Regular Expressions* - RegEx), atuando assim como um serviço de filtragem inicial para proteção contra ameaças de SQLi, a fim de atender aspectos de tempo de resposta e escalabilidade. Portando, o DEA-SQLi tem por objetivo detectar se uma dada consulta possui uma estrutura que indique uma tentativa de SQLi, ou seja, busca filtrar essas possíveis ameaças evitando que soluções de detecção de SQLi mais complexas (que tem um alto tempo de resposta e usam um grande montante de recursos computacionais) sejam executadas sem necessidade.

O DEA-SQLi possui três características base: (A) Definição de uma API para disponibilizar o serviço de detecção de ameaças de SQLi de forma estruturada e interoperável entre os diversos serviços de computação urbana; (B) Execução do processo de detecção de ameaça em ambos os ambientes de Névoa (Borda) ou Nuvem, i.e., considerando um cenário com recursos computacionais restritos de computação em borda, e com a possibilidade de comunicação mais próxima dos usuários, bem como em um ambiente próximo aos serviços de computação urbana e com disponibilidade de recursos computacionais; e, (C) O conjunto de RegEx a ser aplicado é flexível, podendo ser ajustado de acordo com o perfil de ameaças que o serviço de computação urbana precisa para ser protegido. Com relação as RegEx, estas definem cadeias que são validadas de acordo com certas limitações expressas, podendo servir para validar um e-email, CPF, Número de Identidade, entre outros dados de cadastro de usuários [Soewito et al. 2018]. Sendo assim, é possível utilizá-las também para verificar quando algum usuário ou sistema malicioso tenta injetar SQL em algum campo de entrada de dados ou em uma transmissão, durante a sua interação com o sistema, podendo ser em algum campo de preenchimento de login, senha, ou em algum *payload* infectado.

Os resultados dos experimentos, utilizando um conjunto de dados real, sugerem que a solução proposta possui uma eficiência adequada de detecção de ameaças de SQLi, enquanto entrega um tempo de resposta que atende à requisitos de serviços de computação urbana, onde vários cenários foram avaliados, variando o tamanho do banco de dados de RegEx e a carga solicitada na detecção de SQLi. Adicionalmente, os resultados são comparados com a utilização de métodos de aprendizado de máquina, visando traçar um paralelo com outras abordagens existentes.

A solução proposta foi desenvolvida dentro do escopo do projeto de Pesquisa e Desenvolvimento entre UECE e LACNIC[1], que visa desenvolver soluções industriais para proteção de dados sensíveis na Internet, e detectar SQLi é um dos aspectos considerados no processo de proteção, pois compromete a privacidade dos usuários de serviços de

[1] programafrida.net/en/archivos/project/sistema-de-proteccion-de-datos-basado-en-tecnicas-de-anonimizacion-y-que- cumple-con-las-leyes-de-privacidad

computação urbana.

O restante deste artigo está organizado da seguinte forma: a Seção 2 descreve os trabalhos existentes relacionados a SQLi em sistemas gerais e no contexto de computação urbana; a Seção 3 detalha a solução de detecção SQLi proposta; a Seção 4 apresenta o resultado dos experimentos realizados; por fim, a Seção 5 conclui o artigo e cita os trabalhos futuros.

2. Trabalhos Relacionados

Esta seção descreve os principais trabalhos relacionados, e recentemente publicados pela comunidade científica, sobre detecção de SQLi no contexto de computação urbana e sistemas computacionais, incluindo aspectos de desempenho e qualidade de serviço.

Parashar et al. [Parashar et al. 2021] apresentam uma abordagem para identificar informações sobre SQLi em Aplicações de Tecnologia da Informação, usando sumarização de texto a fim de processar os dados, independente de tamanho da entrada. A partir desta sumarização é criado um modelo de aprendizado de máquina supervisionado para automatizar a classificação de SQLi. Contudo, a abordagem dos autores de aplicar duas etapas de processamento, sendo uma delas com aprendizado de máquina, aumenta o tempo de resposta e, consequentemente, diminui a escalabilidade da solução.

Tang et al. [Tang et al. 2020] descrevem um método de detecção de SQLi baseado em redes neurais, visando alcançar altas taxas de acurácia (cerca de 99%). Os autores conduziram as pesquisas estatísticas sobre dados normais e dados de SQLi oriundos dos diários de acesso às URLs de provedores de Internet. Com base nos resultados estatísticos, os autores projetaram oito tipos de recursos e treinaram um modelo MLP. Similarmente, Li et al. [Li et al. 2019] propõem um modelo de floresta profunda (*deep forest*) baseado no algoritmo AdaBoost para detectar os ataques de SQLi complexos, onde a entrada de cada camada é concatenada pelo vetor brutos de características e a média das saídas anteriores, utilizando a taxa de erro para atualizar os pesos em cada camada. Como consequência, no processo de treinamento diferentes recursos são atribuídos com pesos diferentes com base em sua influência no resultado. Ambos os modelos propostos se mostraram eficientes na detecção de SQLi, mas com alto custo computacional, resultando em um tempo de processamento maior do que o ideal para serviços de computação urbana.

Fadolalkarim et al. [Fadolalkarim et al. 2020] apresentam o AD-PROM, um sistema de detecção de anomalias que visa proteger bancos de dados relacionais contra agentes maliciosos. O AD-PROM rastreia as chamadas de sistema executadas pelos programas em execução no sistema computacional que hospeda o banco de dados a ser protegido. A partir deste rastreamento, o AD-PROM analisa os fluxos de controle e dados dos programas e constrói um Modelo oculto de Markov (Hidden Markov Model - HMM) que detecta as anomalias que indicam SQLi que estejam ocorrendo no *host* do banco de dados. Da mesma forma, Xie et al. [Xie et al. 2019] descrevem um método para detecção de SQLi baseado em Redes neurais Convolucionais de Pooling elástico (Elastic-Pooling Convolutional neural networks - EP-CNN). Este método produz uma matriz bidimensional fixa sem truncar dados e detecta efetivamente o SQLi de aplicações web através de correspondência irregulares de características.

Kuroki et al. [Kuroki et al. 2020] propõem um método para identificar a intenção

de SQLi em Firewalls Web, a fim de encurtar o tempo de análise na situação em que informações limitadas estão disponíveis. O método proposto analisa apenas uma consulta SQL parcial em uma solicitação HTTP e estima sua intenção por análise e emulação de sintaxe. Apesar de aplicar uma abordagem de identificar possíveis tentativas de SQLi, este método proposto possui pouca flexibilidade de análise (visto que não preve a possibilidade de ajustar a análise de acordo com o possível perfil de SQLi) e possui pouca eficiência na detecção de SQLi (alcançando cerca de 70% de acurácia).

A partir do levantamento bibliográfico realizado, nota-se que nenhum artigo da literatura se concentrou no desenvolvimento de uma solução escalável para detecção de ameaças SQLi, que é o foco deste artigo. Nossa proposta ao utilizar um conjunto flexível e adaptável de RegEx a serem analisadas atende a expectativa de eficiência na detecção, enquanto atende os aspectos de escalabilidade dos serviços de computação urbana.

3. Proposta

Esta seção detalha a proposta deste artigo, intitulada DEA-SQLi, onde os seguintes tópicos serão descritos: A Seção 3.1 discute as características das ameaças de SQLi, servindo como uma fundamentação para a definição do DEA-SQLi; A Seção 3.2 projeta a arquitetura do DEA-SQLi, apresentando seus módulos e fluxo de funcionamento; A Seção 3.3 descreve os aspectos de implementação do DEA-SQLi; por fim, a Seção 3.4 apresenta como as RegEx aplicadas neste artigo foram formuladas, descrevendo a estruturação de cada cadeia e seu objetivo na detecção de ameaças de SQLi.

3.1. Estrutura dos Ataques SQLi

SQLi é um ataque que explora vulnerabilidades em consultas que acessam os dados de forma insegura, enganando o sistema através de algumas técnicas que expõem dados de usuários cadastrados e autenticados [Kumar and Pateriya 2012]. Existem vários tipos de SQLi, onde pode-se mapear esses tipos da seguinte forma [Lages and Pereira 2022, Yunus et al. 2018, Kumar and Pateriya 2012]:

- Tautologias: As cláusulas ”where”des instruções SQL são alteradas usando termos tautológicos para obter informações sobre o banco de dados de acordo com o resultado da entrada.
- Consultas Logicamente Incorretas: A instrução SQL submetida é errada ou incompleta, buscando adquirir informações sobre o esquema do banco a partir do retorno dessa instrução.
- Consultas de União: A instrução SQL é composta de operações em “union select”, que injetam consultas maliciosas em conjunto com a consulta segura original.
- Ataques Baseados em Inferência: A instrução SQL tenta obter informações do banco de dados a partir de uma estrutura de consulta binária (verdadeiro ou falso) ou através da medição do tempo de retorno de certa instrução.
- Codificações Alternativas: A instrução SQL é alterada usado alguma codificação (tais como hexadecimal, ASCII e Unicode), resultando em uma possível falha em verificadores que buscam por caracteres inválidos que geram injeções de SQL.

Uma característica importante dos SQLi é o uso de campos de usuário e senha como meio para a execução de comandos não autorizados, entretanto, é possível que dispositivos IoT sejam usados como vetor de execução de SQLi, principalmente com a

expansão dos serviços de computação Urbana [Rizvi et al. 2018]. Portanto, faz-se necessário entender a forma que as instruções SQL podem ser realizadas a fim de gerar SQLi, onde cada tipo apresentado possui um perfil de quebra de segurança diferente. A proposta deste trabalho visa englobar a análise destas possibilidades de SQLi de forma rápida e eficiente, aplicando RegEx mapeadas de acordo com o perfil de ameaças que o serviço de computação urbana.

3.2. Arquitetura do DEA-SQLi

A solução proposta é composta por três módulos: *API*, *Análise* e *Atualização*. Além disso, existem duas formas de comunicação: uma comunicação interna direta entre os módulos e uma comunicação externa via Internet que ocorre via a API definida. Uma visão geral do DEA-SQLi e do contexto de implantação é ilustrada na Figura 1.

O módulo de *Análise* executa o processo de detecção de ameaça aplicando um conjunto de RegEx, enquanto que o módulo *Atualização* gerencia o conjunto de RegEx a ser aplicado, onde o conjunto total é armazenado no banco de dados de RegEx na Nuvem (chamado *RegEx BD*). Por fim, a *API* permite que as funcionalidades do DEA-SQLi sejam acessadas pelos diversos serviços de computação urbana, bem como a interação entre os módulos no modelo de funcionamento entre borda e nuvem.

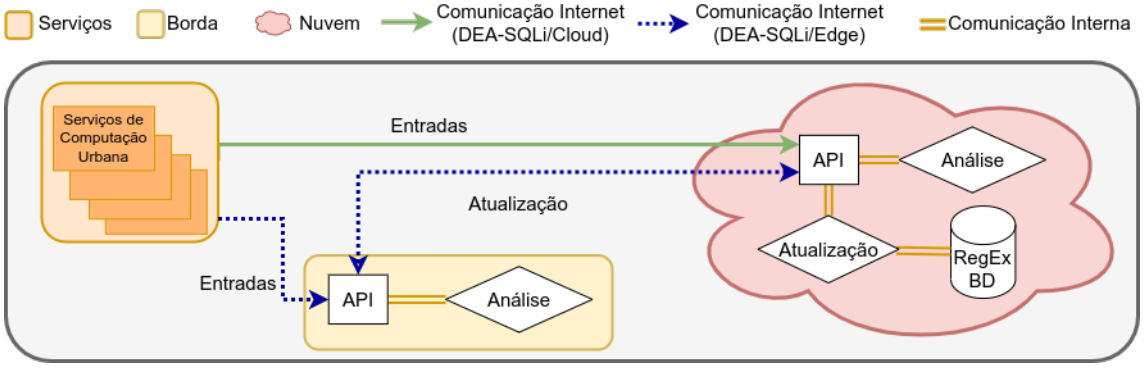

Figura 1. Visão Geral do DEA-SQLi

O DEA-SQLi possui dois modos de funcionamento para detecção de ameaça de SQLi: (I) Serviço de borda da rede (DEA-SQLi/Edge) ou (II) Serviço hospedado em nuvem (DEA-SQLi/Cloud). No DEA-SQLi/Edge há somente o módulo de Análise e a API na Borda, ou seja, neste caso a funcionalidade de detecção de ameaças de SQLi ocorre mais próxima dos serviços de computação urbana, onde a atualização do conjunto de RegEx a ser usado pelo DEA-SQLi será feita através da API com o ambiente de nuvem que hospeda o RegEx BD. Por outro lado, no DEA-SQLi/Cloud tanto a análise das entradas quanto o processo de atualização ocorre através de comunicação interna entre os módulos, sendo assim neste cenário os serviços de computação urbana se comunicam diretamente com a API no ambiente de nuvem.

3.3. Implementação do DEA-SQLi

A solução proposta foi desenvolvida em Python utilizando o micro-framework Flask e *Docker*, com intuito de detectar possíveis tentativas de ataques SQL *Injection* utilizando as Expressões Regulares. Foi desenvolvida utilizando o padrão *Model-View-Controller* (MVC), além de utilizar um banco de dados contendo Expressões Regulares

que serão testadas, a fim de encontrar tentativas de ataques em requisições de texto, dentro de qualquer sistema de informação.

Inicialmente são criados dois contêineres *Docker* para englobar as dependências e as configurações para execução da API e do banco de dados de RegEx (BD RegEx). O contêiner do BD RegEx é utilizado para fazer modificações de estado e consultas das expressões regulares, além de manter os dados com as tentativas de SQLi. No que se refere a API, esta define rotas que envolvem todas as operações de persistência *(Create, Read, Update, Delete - CRUD)* necessárias para manipulação e atualização das expressões regulares no BD RegEx, bem como a rota de detecção que realiza a função de detectar se a entrada em questão é uma ameaça de SQLi. Esta função retorna as expressões regulares cadastradas no banco e executa a expressão regular para a cadeia passada, que no caso serão as ocorrências da base de dados de SQLi. Caso seja correspondente ou não, é retornada uma mensagem indicando.

3.4. Expressões Regulares do DEA-SQLi

O objetivo das expressões regulares no DEA-SQLi é identificar padrões e características comumente presentes em cadeias de caracteres que realizam SQLi. Dado o contexto deste trabalho, sendo um filtro inicial para ameaças de SQLi, as expressões regulares também podem ser ativadas com consultas SQL convencionais, visto que neste contexto consultas SQL não são esperadas, i.e, campos de captura de texto ou *payload* de transmissões heterogêneos. Portanto, faz-se necessário entender a forma que o SQLi é feito e sua estruturação (como discutido na Seção 3.1), onde foram usadas como base as seguintes referências sobre SQLi [Mookhey and Burghate 2004, Lages and Pereira 2022, Yunus et al. 2018, Kumar and Pateriya 2012, Rizvi et al. 2018, M and H B 2022]. Desta forma, foram definidas 8 expressões regulares a serem usadas neste trabalho, as quais são detalhadas a seguir:

1. Detecção dos principais meta-caracteres SQL: Esta expressão regular é usada para detectar possíveis tentativas de SQLi procurando o equivalente hexadecimal das aspas simples, as próprias aspas simples ou a presença do traço duplo, que são indicativos de início de comentário. `(\%27)|(\')|(--[^\r\n]*)|(;%00)`
   - `(\%27)`: Sequência de caracteres `\%27`, que é a forma codificada de URL do caractere de aspas simples `(')`.
   - `(\')`: Próprio caractere de aspas simples `(')`.
   - `(--[\^\r\n]*)`: Sequência de comentários que começa com dois hífens '−−' seguidos por quaisquer caracteres que não sejam quebras de linha.
   - `(;%00)`: Caractere ponto-e-vírgula (;) seguido pelo caractere de byte nulo `(%00)`.
2. Detecção de meta-caracteres SQL secundários: Esta expressão regular é usada para detectar possíveis SQLi procurando sinais de igual, aspas simples, traços duplos, ponto-e-vírgula e o caractere nulo. `((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))`
   - `((%3D)|(=))`: Caractere de sinal de igual (=) ou à versão codificada em URL dele `(%3D)`. Esses caracteres costumam ser usados em consultas SQL para denotar igualdade em comparações.
   - `[\^\n]*`: Qualquer número de caracteres que não sejam um caractere de nova linha `(\n)`.

- `((%27)|('))`: Forma codificada de URL do caractere de aspas simples `(%27)` ou ao caractere de aspas simples `(')`.
- `(--)`: Sequência de hífen duplo (–), que pode ser usada para comentar o restante de uma consulta SQL.
- `((\%3B)|(;))`: Caractere de ponto-e-vírgula (;) ou à versão codificada em URL dele (%3B), que pode ser usada para separar várias instruções SQL em um único consulta.

3. Detecção da cadeia ”or”dentro das strings: Esta expressão regular é usada para detectar possíveis SQLi procurando a palavra 'ou' no formato codificado por URL. `((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))`
   - `((\%27)|(\'))`: Uma aspa simples codificada por porcentagem `(%27)` ou a uma aspa simples simples `(')`
   - `((\%6F)|o|(\%4F))`: Uma letra minúscula 'o' codificada por porcentagem `(%6F)`, uma letra minúscula simples 'o' ou uma letra maiúscula codificada por porcentagem 'O' `(%4F)`
   - `((\%72)|r|(\%52))`: Uma letra minúscula 'r' codificada por porcentagem `(%72)`, uma letra minúscula simples 'r' ou uma letra maiúscula 'R' codificada por porcentagem `(%52)`
4. RegEx para busca de cadeias com operadores lógicos: Esta expressão regular é usada para detectar possíveis SQLi procurando as palavras 'and' ou 'or' seguidas por um ou mais espaços. `(\W)(and|or)\s*`
   - `(\W)`: Caractere que não seja uma palavra (ou seja, um caractere que não seja uma letra, dígito ou sublinhado).
   - `(and|or)`: Palavra 'and' ou 'or'.
   - `\s*`: Zero ou mais caracteres de espaço em branco.
5. RegEx para buscar 'UNION': Esta expressão regular é usada para detectar possíveis SQLi com a palavra 'union' e no formato codificado por URL. `((\%27)|(\'))UNION`
   - `((\%27)|(\')`: Aspa simples codificada por porcentagem `(%27)` ou a uma aspa simples simples `(\')`.
   - `UNION`: Palavra 'UNION' em letras maiúsculas.
6. RegEx para *queries*: Essa expressão regular é usada para detectar possíveis SQLi pesquisando palavras-chave SQL como 'select', 'drop', 'insert', 'delete', 'update', 'create' e 'alter'. `([\s\(\)])*(select|drop|insert|delete|update|create|alter)([\s\(\)])*`
   - `([\s\(\)])*`: Esta parte corresponde a zero ou mais ocorrências de qualquer caractere de espaço em branco `(\s)` ou parênteses esquerdo ou direito.
   - `(select|drop|insert|delete|update|create|alter)`: Uma das palavras-chave SQL 'select', 'drop', 'insert', 'delete', 'update', 'create' ou 'alter'.
7. RegEx para 'exec' e 'execute': Esta expressão regular é usada para detectar possíveis SQLi pesquisando as palavras-chave SQL 'exec' ou 'execute'. `([\s\(\)])*(exec|execute)([\s\(\)])*`
   - `([\s\(\)])*`: Zero ou mais ocorrências de qualquer caractere de espaço em branco `(\s)` ou parênteses esquerdo ou direito.
   - `(exec|execute)`: Uma das palavras-chave SQL 'exec' ou 'execute'.
8. RegEx para *queries* com *AND*: Essa expressão regular é usada para detectar possíveis SQLi pesquisando a palavra 'e'(*AND*) em vários formatos, como formato codificado por URL, símbolo mais e operadores lógicos. `(\%20and|\+and|&&|\&\&)`

- `(\%20and)`: onde `%20` representa um caractere de espaço codificado por URL.
- `(+and)`: onde + representa um caractere de espaço codificado por URL no contexto de um parâmetro de consulta.
- `(&&)` e `(\&\&)`: o operador e lógico na maioria das linguagens e ambientes de programação.

O conjunto de RegEx apresentado visa atender os diferentes aspectos de estruturação de ataques SQLi, como apresentado na Seção 3.1. Assim, pretende-se englobar a maior quantidade de ameaças possíveis, maximizando assim o número de detecção de ameaças de SQLi. Por fim, é válido ressaltar que este conjunto de 8 RegEx foi modelado como uma proposta base para este trabalhando, sendo possível atualizar este conjunto de RegEx dinamicamente no DEA-SQLi, como descrito na Seção 3.2.

4. Experimentos

Esta seção apresenta os experimentos realizados para avaliar o desempenho da solução proposta para detecção de ameaças de SQLi, que está disponível no repositório[2], ou seja, a ideia dos experimentos realizados é mostrar a viabilidade do DEA-SQLi em um ambiente real. Para a realização dos experimentos, foi definido um cenário realista com informações de um conjunto de dados de SQLi e diversos ambientes computacionais (máquinas virtuais em provedor de nuvem e máquinas físicas), possibilitando uma avaliação adequada da solução e seu impacto. A subseção 4.1 apresenta a configuração dos experimentos, enquanto a subseção 4.2 discute os resultados obtidos.

4.1. Configuração dos Experimentos

Os ambientes de teste usados nos experimentos consideraram tanto o uso de máquinas físicas quanto um ambiente de computação em nuvem. Como ambiente de nuvem foi usado a Huawei Cloud[3] com diversos tipos de máquinas virtuais Elastic Cloud Server (ECS)[4]. É válido ressaltar que os ECS no Huawei Cloud foram configurados com 4 CPUs Virtuais (vCPUs) e 16Gb de memória. Assim, foram considerados os seguintes casos:

- Huawei Cloud - General Computing Plus (GCP - versão C3): Fornece um equilíbrio de recursos de computação, memória e rede, bem como um nível básico de desempenho de vCPU com a capacidade de dar um *burst* acima da linha de base.
- Huawei Cloud - Memory Optimized (MO - versão M3): Possui alto desempenho de acesso a memória, sendo projetada para aplicações com uso intensivo de memória que processam grandes volumes de dados.
- Huawei Cloud - High-Performance Computing (HPC - versão H1): Adequada para aplicações que exigem um grande número de recursos de computação paralela e serviços de infraestrutura de alto desempenho para atender aos requisitos de computação e armazenamento, bem como garantir alta eficiência de renderização.
- Huawei Cloud - Kunpeng (KPG - versão KC1): Usa um processador Kunpeng 920 para, fornecer de maneira econômica, um nível de linha de base de desempenho

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

[3]huaweicloud.com

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

de vCPU com a capacidade de atender aos requisitos de migração de serviços de infraestrutura para a nuvem.
- Physical Machine (PM): Máquina de uso diário comum para tarefas gerais, composta de um processador CPU Intel Core i5-12400, 8GB DDR4 2666MHz de memória e um disco SSD de 256GB.

Os experimentos foram baseados em um conjunto de dados de SQLi[5] aplicado em diversos trabalhos na literatura [Rahul et al. 2021, Devalla et al. 2022, Hosam et al. 2021, Roy et al. 2022]. Este conjunto de dados possui 19340 entradas, sendo 7962 delas entradas benignas, enquanto que 11378 delas são SQLi. Assim, pode-se ter uma avaliação completa em relação a proposta.

A capacidade do DEA-SQLi detectar corretamente as ameaças de SQLi é comparada com o uso de técnicas de aprendizado de máquina[6]: Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), Naive Bayes (NB), Decision Tree (DT), Convolutional Neural Network (CNN). Adicionalmente, foi avaliado o tempo que o DEA-SQLi leva para fazer os processos de detecção e atualização de RegEx (onde foi utilizado um intervalo de confiança de 95% para 100 experimentos realizados). Assim, foram consideradas as seguintes métricas de avaliação de desempenho [Moreira et al. 2021]:

- Acurácia (em porcentagem): Taxa de classificações corretas de acordo com a Equação 1, ou seja, os casos Verdadeiro Positivo (VP) e Verdadeiro Negativo (VN) em relação a todos os outros casos (VP, VN, Falso Positivo - FP e Falso Negativo - FN).
$$Acuracia = \frac{VP + VN}{\mathrm{VP} + FN + FP + VN} \tag{1}$$
- Recall (em porcentagem): Eficiência em detectar corretamente a entrada analisada, ou seja, a taxa de VP em relação ao total de casos positivos ($VP + FN$). Portanto, o Recall é definido na Equação 2.
$$Recall = \frac{\mathrm{VP}}{\mathrm{VP} + FN} \tag{2}$$
- Precisão (em porcentagem): Determina a capacidade de acertar quais das detecções positivas realmente são positivas, seguindo a definição expressa na Equação 3.
$$Precisao = \frac{\mathrm{VP}}{\mathrm{VP} + FP} \tag{3}$$
- Tempo de Análise (em milissegundos): tempo necessário para analisar uma entrada para definir se esta é uma ameaça de SQLi ou não.
- Tempo de Atualização (em milissegundos): tempo gasto pela solução para atualizar o conjunto de RegEx a ser usado durante o processo de detecção de ameaças de SQLi. É válido ressaltar que este tempo não considera o tempo de comunicação de Internet, ou seja, é a soma do tempo de extrair as RegEx a serem atualizadas da base de dados de RegEx, e do Analisador carregar esse novo conjunto para poder ser usado no processo de detecção.

[5]kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset
[6]kaggle.com/code/chinonsocynthia/sql-inject-using-linear-models-and-cnn

4.2. Resultados

Esta subseção discute os resultados dos experimentos realizados, onde a Figura 2 apresenta o tempo de processamento da solução para determinar se a entrada em questão é uma ameaça de SQLi ou não, enquanto que a Figura 3 ilustra o tempo de atualização do conjunto de RegEx usados no processo de detecção de ameaças de SQLi. Por fim, a Tabela 1 mostra os resultados de Acurácia e Recall, comparado a eficiência do DEA-SQLi com as técnicas de aprendizado de máquina.

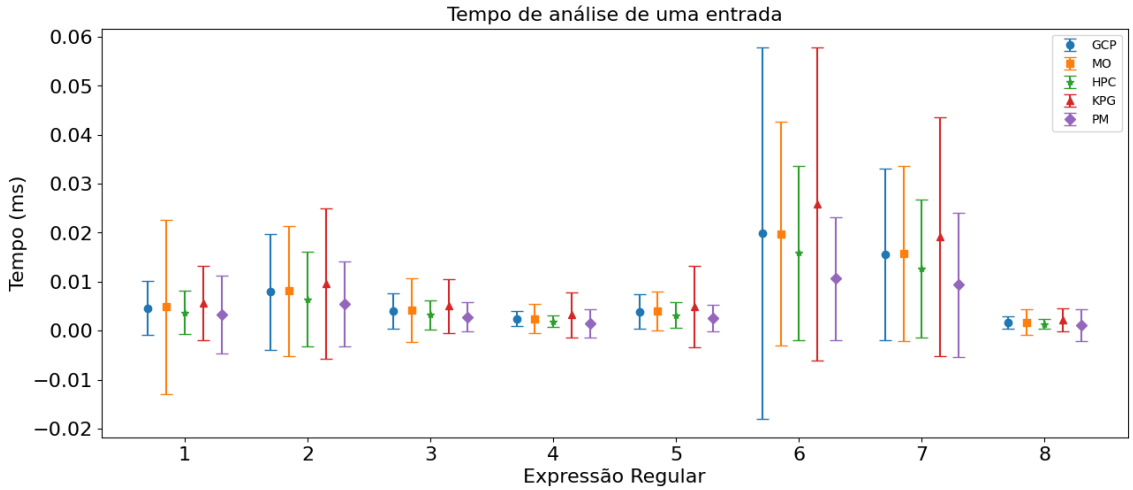

Figura 2. Tempo de Análise

Conforme observado na Figura 2, o comportamento do tempo de análise de uma entrada é similar independente do ambiente computacional utilizado (máquinas virtuais em nuvem ou máquina física na borda). Contudo, percebe-se dois aspectos: (i) A máquina física (PM) possui uma maior variação no tempo de análise, devido ao maior número de processos e funcionalidades que executam em conjunto com o processo de análise do DEA-SQLi, fato que não acontece com as máquinas virtuais em nuvem (GCP, MO, HPC e KPG) que atuam de forma dedicada ao DEA-SQLi; e, (ii) as RegEx 1 e 5 tendem a ter um tempo de processamento maior, devido a sua estrutura mais robusta, como discutido na Seção 3.4. Portanto, é possível concluir, no que se refere ao tempo de análise, que o DEA-SQLi possui um tempo adequado em relação ao que se espera dos serviços de computação urbana, bem como consegue atender aspectos de escalabilidade, visto que este é capaz de analisar 1000 requisições em cerca de 10 milissegundos, tempo pequeno quando comparado com a comunicação dos dados via Internet [M and H B 2022, Rizvi et al. 2018].

Com relação aos resultados do tempo de atualização do conjunto de RegEx, mostrados na Figura 3, foi variado o tamanho do conjunto de RegEx a serem atualizados, possibilitando uma análise sobre a capacidade do DEA-SQLi de manter a escalabilidade de solução quando o conjunto de RegEx aplicado cresce. Nota-se que o desempenho da máquina física (PM) é melhor quando comparado as máquinas virtuais em nuvem (GCP, MO, HPC e KPG), alcançando um tempo de atualização médio cerca de 30% menor. É válido ressaltar que durante o processo de atualização as RegEx são resgatas da base de dados que fica em disco, sendo que a velocidade de acesso ao disco na PM é maior que no ambiente de nuvem.

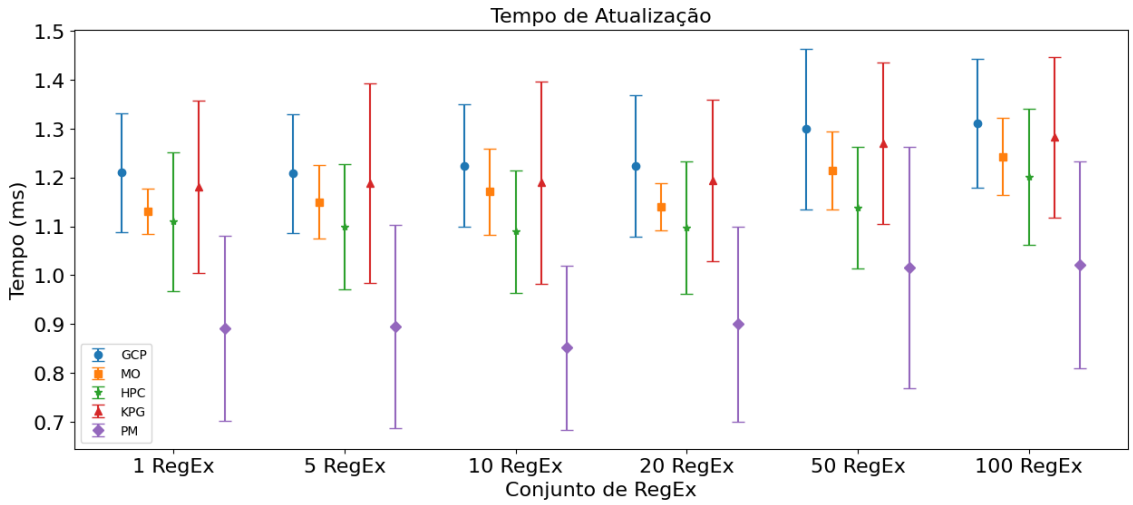

Figura 3. Tempo de Atualização do Conjunto de RegEx

Tabela 1. Resultados de Eficiência

Métrica	LR	RF	SVM	NB	DT	CNN	DEA-SQLi
Precisão (%)	0.71	0.99	0.20	0.99	0.99	0.99	0.99
Recall (%)	0.98	0.72	0.99	0.92	0.61	0.90	0.92
Acurácia (%)	0.92	0.90	0.79	0.97	0.84	0.97	0.95

Os dados da Tabela 1 representam a capacidade das soluções avaliadas em detectar que uma determinada entrada é um SQLi ou não. No caso do DEA-SQLi, estes resultados determinam a eficiência deste em detectar uma possível ameaça e esta ser realmente uma ameaça que precisa ser analisada por soluções mais complexas, possibilitando uma economia de tempo e recursos computacionais para os serviços de computação urbana. A partir destes dados, é possível identificar que a eficiência do DEA-SQLi é nivelada com o uso das técnicas de aprendizado de máquina, sendo que este ainda supera algumas técnicas no que se refere as métricas de desempenho analisadas. Este fato possibilita assim que esta seja aplicada uma primeira camada de proteção, filtrando as entradas para determinar se algo pode ser uma ameaça de SQLi.

5. Conclusão

Os serviços de Computação Urbana se tornaram realidade nos últimos anos devido a expansão das tecnologias de comunicação e informação pelas cidades e centros metropolitanos. Estes serviços geram um volume de dados massivo e com características heterogêneas, que, após o processo de coleta por sensoriamento urbano, são armazenados em banco de dados relacionais. Este cenário impulsionou as ameaças em relação a esses banco de dados, dentre essas ameaças os ataques de SQLi. A fim de lidar com esta realidade, este artigo apresentou o DEA-SQLi, uma solução de detecção de ameaças de SQLi baseado em RegEx que tem por objetivo atuar como um serviço de filtragem inicial para proteção contra ameaças de SQLi a fim de atender aspectos de tempo de resposta e escalabilidade.

Como trabalho futuro, pretende-se desenvolver um modelo de Inteligência Artificial que consiga ser integrado ao processo de análise via RegEx, possibilitando o agru-

pamento e classificação do SQLi de acordo com o perfil, ampliando assim a capacidade de solução de atender os requisitos dos serviços de computação urbana. Adicionalmente, tem-se por objetivo evoluir a análise das entradas expandindo as possibilidades de RegEx a serem utilizadas.

Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (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

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.

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.

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.

Kumar, P. and Pateriya, R. (2012). A survey on sql injection attacks, detection and prevention techniques. In *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*, pages 1–5.

Kuroki, K., Kanemoto, Y., Aoki, K., Noguchi, Y., and Nishigaki, M. (2020). Attack intention estimation based on syntax analysis and dynamic analysis for sql injection. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1510–1515.

Lages, G. and Pereira, R. (2022). Estudo comparativo entre tecnicas de deteccao e prevencao de ataques de injecao sql. In *Anais do XVII Escola Regional de Banco de Dados*, pages 135–138, Porto Alegre, RS, Brasil. SBC.

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.

Mookhey, K. and Burghate, N. (2004). Detection of sql injection and cross-site scripting attacks. *Symantec SecurityFocus*.

Moreira, D. A. B., Marques, H. P., Costa, W. L., Celestino, J., Gomes, R. L., and Nogueira, M. (2021). Anomaly detection in smart environments using ai over fog and cloud computing. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–2.

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

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.

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.

Rizvi, S., Kurtz, A., Pfeffer, J., and Rizvi, M. (2018). Securing the internet of things (iot): A security taxonomy for iot. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 163–168.

Rodrigues, D. O., Santos, F. A., Filho, G. P. R., Akabane, A. T., Cabral, R., Immich, R., Junior, W. L., Cunha, F. D., Guidoni, D. L., Silva, T. H., Rosário, D., Cerqueira, E., Loureiro, A. A. F., and Villas, L. A. (2019). Computação urbana da teoria à prática: Fundamentos, aplicações e desafios.

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.

Soewito, B., Gunawan, F. E., Hirzi, and Frumentius (2018). Prevention structured query language injection using regular expression and escape string. *Procedia Computer Science*, 135:678–687. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

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.