Análise de diferentes técnicas de pré-processamento em algoritmos de Aprendizado de Máquina na detecção de SQL *Injection*

Vanessa C. O. Souza[1], **Erick T. A. Silva**[1], **Rafael M. D.** [1], **Melise M. V. Paula**[1]

[1]Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
Caixa Postal 50 – 37500-903 – Itajubá – MG – Brasil

{vanessasouza, frinhani, melise}@unifei.edu.br, erickthomas_alves@hotmail.com

Abstract. *This work aimed to use machine learning algorithms to detect SQL Injection from five different input data treatment methods. Using the Naive Bayes, Support Vector Machine (SVM), Gradient Boosting Tree (GBT), and Random Forest (RF) algorithms, the results point to the superiority of the GBT and RF ensemble algorithms. GBT obtained the best result with the set of metrics G-Test and Entropy, calculated on tokenization and transformation with regular expression, presenting an accuracy of 98,46%. The Naive Bayes algorithm presented the worst performance in all evaluated sets.*

Resumo. *Atualmente, a SQL Injection é uma das maiores ameaças à segurança das aplicações WEB e, por isso, diversas abordagens vêm sendo analisadas para tentar resolver esse problema. O objetivo deste trabalho foi utilizar algoritmos de aprendizado de máquina para detectar SQL Injection a partir do tratamento dos dados de entrada de cinco formas diferentes, variando a tokenização, transformação e extração de atributos das bases de SQL Padrão e Injection. Os algoritmos utilizados foram Naive Bayes, Support Vector Machine (SVM), Gradient Boosting Tree (GBT) e Random Forest (RF). O melhor resultado foi obtido com o GBT com as métricas G-Test e Entropia, calculadas sobre tokenização e transformação com expressão regular, apresentando acurácia de 98,46%.*

1. Introdução

Com a evolução das tecnologias associadas à Internet, aplicações web tornaram-se essenciais nas atividades corriqueiras da sociedade, tais como comunicação, comércio eletrônico e transações financeiras. Em função do valor associado e confidencialidade das informações tratadas pelas aplicações web, elas são alvos constantes de ataques por motivações comerciais, financeiras e/ou ideológicas. Em 2020, a SQL *Injection* ocupou o topo da lista de vulnerabilidades de aplicações web [OWASP 2020].

O ataque por SQL *Injection* (SQLIA) acontece quando um código malicioso insere (injeta) palavras-chave ou operadores SQL em consultas legítimas realizadas por uma aplicação web a um banco de dados. A origem do SQLIA se dá, portanto, pela validação inadequada da entrada do usuário. Se o ataque for bem sucedido, usuários não autorizados acessam e manipulam o banco de dados da organização. Esta violação de segurança pode acarretar roubo de informações, perda de consistência e indisponibilidade do servidor [Fang et al. 2018, Hanmanthu et al. 2015, Rankothge et al. 2020].

Segundo [Tang et al. 2020], ataques SQLIA causam perdas financeiras em pequenas e grandes empresas, além do vazamento de dados sensíveis dos usuários. Para [Chen et al. 2018], é uma das vulnerabilidades mais prejudiciais na segurança da web e, portanto, sua detecção e prevenção torna-se ainda mais relevante.

Diversas pesquisas propõem identificar e prevenir ataques SQLIA utilizando técnicas de aprendizado de máquina [Rankothge et al. 2020]. A literatura comprova que tais algoritmos são eficientes na detecção de SQLIA [Fang et al. 2018, Joshi and Geetha 2014, Kim and Lee 2014, Mishra 2019, Tang et al. 2020]. No entanto, diferentes formas de realizar o pré-processamento dos dados e diferentes algoritmos de aprendizado de máquina são abordados nesses trabalhos, mas não há uma análise comparativa desses métodos.

O objetivo deste trabalho foi avaliar o impacto de diferentes estratégias de pré-processamento de dados na classificação de SQLIA por algoritmos de aprendizado de máquina. Para tanto, bases de códigos SQL legítimos e *injection* foram pré-processados de cinco formas diferentes e classificados utilizando os algoritmos *Naive Bayes, Support Vector Machine (SVM), Gradient Boosting Tree (GBT)* e *Random Forest (RF)*. O pré-processamento foi baseado nos trabalhos de [Choi et al. 2011], [Joshi and Geetha 2014] e [Mishra 2019].

O restante do trabalho está dividido da seguinte forma: a Seção 2 apresenta uma breve discussão sobre aprendizado de máquina na detecção de SQLIA. Os trabalhos correlatos são apresentados na Seção 3. A Seção 4 apresenta a metodologia. Na Seção 5, são apresentados os resultados e discussões. As conclusões são apresentadas na Seção 6.

2. Aprendizado de máquina na detecção de SQL *Injection*

Esta seção apresenta as etapas do processamento comumente empregadas em trabalhos que utilizam algoritmos de aprendizado de máquina na detecção SQLIA: pré-processamento dos dados, classificação e análise dos resultados.

No pré-processamento, o texto é reduzido a um conjunto de métricas que servirão de entrada para o classificador. A primeira atividade desta etapa é a chamada *"tokenização"*, que consiste em dividir o texto em segmentos menores (*"tokens"*). Para isso, diversas abordagens podem ser utilizadas, como o uso de expressão regular ou o chamado *"blank space"*. No caso da expressão regular, uma lista de palavras e símbolos é utilizada para dividir o texto. Já no *"blank space"*, um *token* é gerado a cada espaço em branco encontrado no texto.

Após a extração dos *tokens*, ocorre a atividade de transformação, cujo objetivo é obter alguma semântica a partir dos *tokens* gerados. Para isso, os *tokens* podem ser quantificados e agrupados, seguindo diferentes critérios.

A Figura 1 apresenta um exemplo de *"tokenização"* e transformação. O código SQL foi dividido em *tokens* utilizando espaços em branco (Figura 1A). Posteriormente, cada tipo de *token* foi quantificado (Figura 1B). Uma tabela de rotulação é utilizada para qualificar os *tokens* (Figura 1C). Posteriormente, *tokens* com mesmo rótulo são agrupados e contabilizados (Figura 1D). Na tokenização por expressão regular, a tabela da Figura 1C representa uma possibilidade de gerar os *tokens*.

Uma alternativa à rotulação dos *tokens* é o uso do N-Gram, que combina

sequências de palavras ou letras de comprimento N [Cavnar and Trenkle 1994]. A literatura de SQLIA normalmente faz uso do 3-grams. A Figura 2 ilustra o conjunto 3-grams gerado para um trecho de código SQL.

A última etapa do pré-processamento dos dados é a extração de atributos, que tem por objetivo representar o texto numericamente, de modo a facilitar a classificação pelos algoritmos de aprendizagem de máquinas [Aggarwal and Zhai 2012].

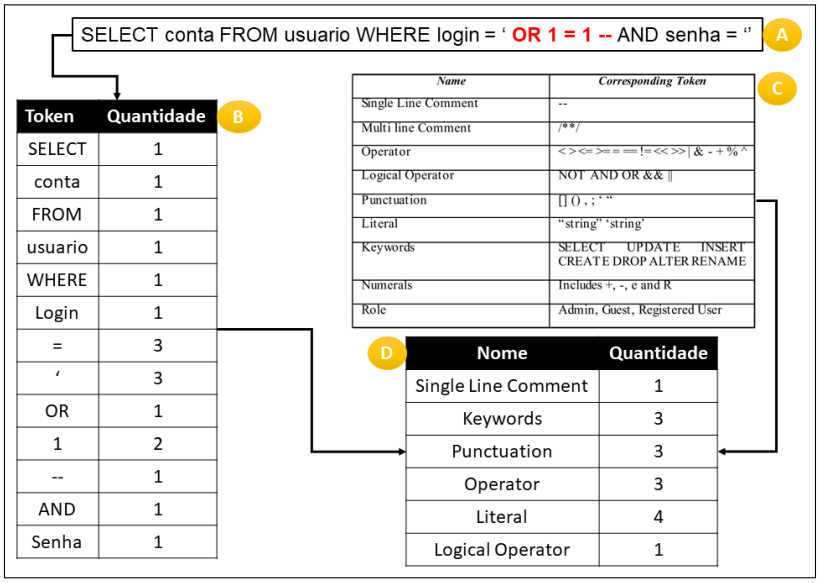

Figura 1. Ilustração do processo de tokenização e rotulagem de dados. Em A), o código SQL recebido pela aplicação. O texto destacado é a injeção SQL; Em B) os tokens gerados; Em C) a tabela de expressão regular e suas classes; Em D) a transformação dos tokens em classes e sua quantificação.

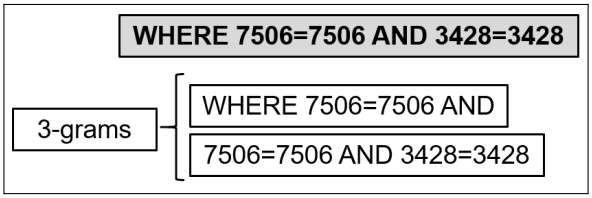

Figura 2. Exemplo de um conjunto de sequências 3-gram sobre um trecho de código SQL.

Na etapa de classificação, os algoritmos de aprendizado de máquina são aplicados ao conjunto de métricas extraídas do texto, e, posteriormente, os resultados são avaliados em relação à acurácia.

3. Trabalhos Correlatos

Mishra [Mishra 2019] desenvolveu um trabalho para detecção de *injections* utilizando os algoritmos de aprendizado de máquina *Gradient Boosting Tree* e *Naïve Bayes*. A autora utilizou uma base de dados de SQL Padrão de quatro mil registros gerados a partir de um formulário web. A base de SQL *Injection* foi gerada com a ferramenta Libinjection, com seis mil registros. O método de tokenização foi expressão regular. Na etapa de extração de atributos foram computadas as medidas quantidade total de *tokens*, G-Test e entropia.

Na etapa de classificação, o algoritmo *Naïve Bayes* apresentou acurácia de 92,8% e o Gradient Boosting Tree, de 97,4%. O trabalho foi desenvolvido em Python.

No trabalho de [Choi et al. 2011], a etapa de pré-processamento envolveu a tokenização com expressão regular e, posteriormente, os dados foram rotulados e agrupados com N-Gram (3-gram). Aplicando o SVM em uma base com 342 registros de SQL Padrão e 472 registros de SQL *Injection*, obteve-se uma acurácia global de 98,04% e baixa taxa de falsos-positivos (0.015). O trabalho foi implementado em Java.

Em [Joshi and Geetha 2014], os autores adaptaram a metodologia de [Choi et al. 2011] para detecção de SQL *Injection*. Para tanto, o processo de tokenização seguiu uma tabela de expressão regular diferente de [Choi et al. 2011]. Os autores também não utilizaram o N-Gram. A base de SQL Padrão continha 101 registros e a *Injection*, 77 registros. O algoritmo de classificação utilizado foi o *Naïve Bayes*. Os autores relatam uma acurácia de 93,3%.

4. Materiais e Métodos

O trabalho foi desenvolvido em Python, versão 3.7, com suporte da plataforma Anaconda[1]. O ambiente de testes foi uma máquina com sistema operacional Windows 10, processador core i5 4670 e 12GB de memória RAM.

A metodologia de mineração de dados proposta por [Nayak and Qiu 2005] foi utilizada para orientar as etapas do processo. Ela é definida em quatro fases: i)Definir os objetivos; ii)Pré-processamento; iii)Modelagem; e iv) Análise dos resultados. O objetivo definido foi obter um modelo de classificação que possa ser aplicado na detecção de SQLIA a partir de registros contendo SQL padrão e SQL *injection*.

4.1. Pré-Processamento dos Dados

A Figura 3 ilustra o processo de Pré-Processamento dos dados que culminou em cinco conjuntos de dados testados na classificação. As etapas são detalhadas a seguir.

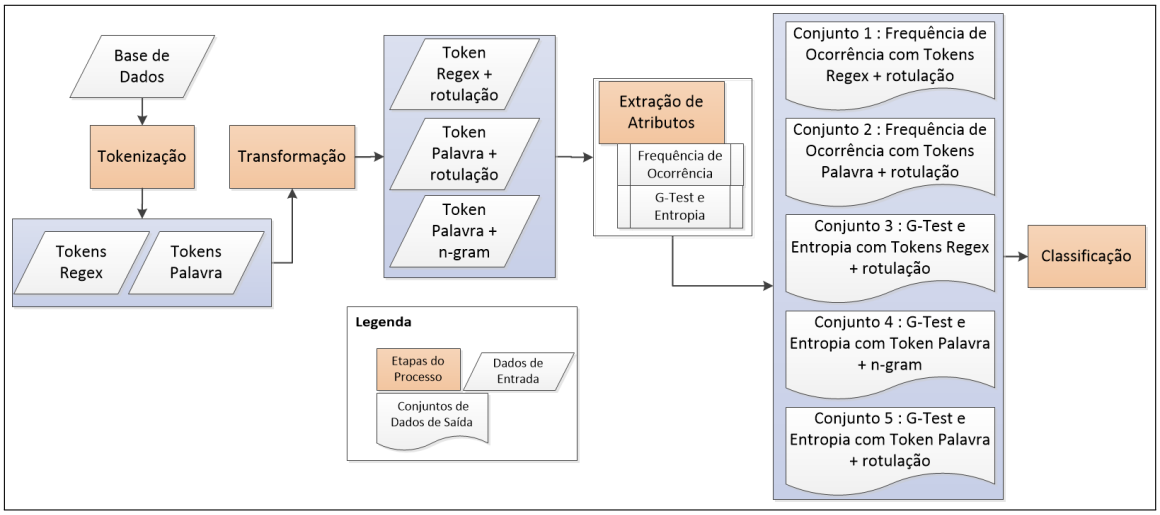

Figura 3. Etapas do pré-processamento dos dados e conjuntos gerados.

[1] https://www.anaconda.com/

- **Base de Dados:** O conjunto com dados SQL *Injection* contém 5924 registros e foi gerado com a ferramenta SQLMap[2]. A base de dados de SQL padrão (sem *Injection*) foi obtida no Github[3], com 6339 registros. A base de SQL padrão foi formatada removendo-se o trecho da consulta até a cláusula WHERE para ficar consistente com a base de *Injection* gerada pelo SQLMap. É importante salientar que a base de SQL *injection* foi gerada com diferentes tipos de *injection*. Ambas as bases apresentam sintaxe SQL do SGBD Oracle.
- **Tokenização:** O processo de tokenização foi realizado de duas formas distintas: expressão regular (regex) e *blank space* (palavras). Como expressão regular, foram selecionados comandos SQL comumente utilizados em SQL *Injection*, considerando o SGBD Oracle[4]. A partir da base de dados de entrada foram gerados dois conjuntos de saída: 'tokens regex', 'tokens palavra'.
- **Transformação:** O conjunto 'tokens palavra' foi transformado utilizando rotulação regex, a partir da coluna Rótulo da Tabela 1; e criando conjunto 3-grams. Já o conjunto 'tokens regex' foi apenas rotulado a partir da Tabela 1.
- **Extração de Atributos:** Considerando os trabalhos de [Joshi and Geetha 2014] e [Mishra 2019], duas estratégias de extração de atributos foram implementadas: frequência de ocorrência; e G-Test e Entropia. A frequência de ocorrência conta a quantidade de *tokens* rotulados em cada categoria, como na Figura 1D. O G-Test e a Entropia são estatísticas, cujos detalhes podem ser vistos em [Mishra 2019]. Para o conjunto n-gram não faz sentido computar a frequência porque as sentenças são diferentes. Portanto, ao final desta etapa, foram gerados cinco conjuntos de dados, avaliados pelos algoritmos de classificação. Em cada conjunto havia ainda a informação da quantidade de *tokens* e o tipo da SQL (padrão ou *injection*).

Portanto, ao final desta etapa cinco conjuntos de dados foram gerados: **Conjunto 1:** Frequência de ocorrência computada sobre *tokens* gerados e rotulados com expressão regular; **Conjunto 2:** Frequência de ocorrência computada sobre *tokens* gerados por *blank space* e rotulados com expressão regular; **Conjunto 3:** G-Test e Entropia computados sobre *tokens* gerados e rotulados com expressão regular; **Conjunto 4:** G-Test e Entropia computados sobre *tokens* gerados por *blank space* e transformados em 3-gram; **Conjunto 5:** G-Test e Entropia computados sobre *tokens* gerados por *blank space* e rotulados com expressão regular.

Tabela 1. Lista de rótulos e comandos SQL utilizados como expressão regular.

Rótulo	Comandos
UNION	UNION SELECT, UNION ALL SELECT
BOOLEAN	=, <=, >=, !=, II, &
DROP	DROP
DATABASE	DATABASES
PREFIX	‘, ‘’, “”, “”, “’
SQLFUNC	Comandos SQL diversos
PLAIN	Qualquer texto não SQL

[2]Versão 1.3.11 - `http://sqlmap.org/`

[3]`https://github.com/jkkummerfeld/text2sql-data/tree/master/data`

[4]`https://docs.oracle.com/cd/E11882_01/server.112/e41085.pdf`

4.2. Modelagem e Análise dos Resultados

A modelagem define os algoritmos que serão utilizados no projeto. Os principais algoritmos utilizados no problema SQLIA são o *Naïve Bayes* [Joshi and Geetha 2014, Lodeiro-Santiago et al. 2017, Mishra 2019] e o SVM [Chen et al. 2018, Choi et al. 2011, McWhirter et al. 2018]. Além desses, [Mishra 2019] utilizou o *Gradient Boosting Tree*. Considerando o bom resultado obtido pela autora com um algoritmo *ensemble*, o *Random Forest* também foi avaliado.

Na etapa de Análise dos Resultados, foram utilizadas as métricas acurácia global dos classificadores e o tempo de processamento (treinamento e classificação). A acurácia dos modelos foi obtida a partir do algoritmo *cross-validation* com 10 pastas.

5. Resultados e Discussões

Na Tabela 2, são apresentados os resultados. Destaca-se em azul a melhor acurácia de cada classificador. Em verde as melhores acurácias para cada conjunto de dados e, em laranja, o pior tempo de processamento. Os algoritmos *ensemble* (*Random Forest* - RF e *Gradient Boosting Tree* - GBT) apresentaram as melhores acurácias. O RF foi melhor para os conjuntos 1 e 2, cuja métrica foi a frequência de ocorrência. Já o GBT foi melhor nos conjuntos 3, 4 e 5, que utilizaram as métricas G-Test e Entropia. Desses, os conjuntos 'token regex' e 'token palavra' com rotulação (conjuntos 3 e 5) tiveram os melhores resultados de classificação, ambos com 98,4% de acerto.

Tabela 2. Acurácia global e tempo de processamento para os cinco conjuntos de dados oriundos do pré-processamento.

Conjunto de Dados	Random Forest		Gradient Boosting Tree		SVM		Naive Bayes	
	Acurácia	Tempo	Acurácia	Tempo	Acurácia	Tempo	Acurácia	Tempo
1) Frequência de ocorrência com token regex e rotulação	92,70%	11,91s	89,90%	19,97s	91,69%	33,94s	81,15%	7,06s
2) Frequência de ocorrência com token palavra e rotulação	94,03%	17,41s	93,70%	20,72s	91,72%	25,23s	81,98%	11,98s
3) G-Test e Entropia com token regex e rotulação	96,09%	72,01s	98,46%	84,09s	91,01%	86,61s	85,48%	56,89s
4) G-Test e Entropia com token palavra e n-gram	94,64%	106,86s	95,88%	105s	88,67%	112,96s	82,21%	101,61s
5) G-Test e Entropia com token palavra e rotulação	95,32%	89,30s	98,49%	87,41s	88,81%	98,99s	84,73%	75,05s

Nesse sentido, a Tabela 2 apresenta alguns resultados importantes desse trabalho:

- O uso das estatísticas G-Test e Entropia mostrou-se superior à frequência de ocorrência para classificação de SQLIA.
- Dentre os conjuntos de dados que utilizaram G-Test e Entropia, o uso do 3-gram levou a um pior desempenho em todos os classificadores.
- Dentre os conjuntos de dados que utilizaram G-Test e Entropia, o 'token regex' e rotulação (conj. 3) apresentou melhor desempenho em todos os classificadores.
- Dentre os conjuntos de dados que utilizaram Frequência de Ocorrência, o 'token palavra' e rotulação (conj.2) apresentou melhor desempenho em todos os classificadores.
- A acurácia do conjunto 4 foi boa, porém o uso do n-gram apresentou maior tempo de processamento em todos os classificadores.
- O algoritmo *Naïve Bayes* apresentou pior desempenho em todos os conjuntos.

Em relação ao tempo de processamento, o conjunto 4 apresentou os piores resultados, com tempos sempre acima de 100 segundos. Os conjuntos 1 e 2 apresentaram os menores tempos de processamento em todos os classificadores. Dentre os conjuntos 3 e 5, que apresentaram as melhores acurácias com o GBT, o conjunto 3 foi processado em tempo inferior ao conjunto 5 (cerca de 17 segundos a menos) .

O conjunto 2 foi baseado no trabalho de [Joshi and Geetha 2014]. Os autores utilizaram o *Naïve Bayes* e relatam acurácia de 93,3%, resultado não alcançado neste trabalho, que foi de 81,98%. Vale destacar que os autores utilizaram outras entradas, como o tipo de permissão do usuário no banco. A base de dados também era menor do que a utilizada neste trabalho.

O conjunto 3 baseou-se no trabalho de [Mishra 2019]. No entanto, neste trabalho o algoritmo *Naïve Bayes* apresentou desempenho pior (85,48%) do que o relatado pela autora, de 92,8% de acurácia. Já o GBT apresentou melhor desempenho (98,46%), uma vez que a autora relata acurácia de 97,4%.

O conjunto 4 teve como base o trabalho de [Choi et al. 2011]. Os autores relatam acurácia de 98,04% com o SVM, superior a encontrada neste trabalho (88,67%). A base de dados também era menor do que a utilizada neste trabalho.

Os conjuntos 1 e 5 foram variações derivadas dos trabalhos acima mencionados. Verifica-se que o conjunto 5 apresentou uma acurácia idêntica a do conjunto 3, mas consome mais tempo de processamento.

6. Considerações Finais

Neste trabalho foram avaliadas cinco formas de pré-processamento de dados SQLIA, com objetivo de identificar *injection* por meio de algoritmos de aprendizado de máquina. A principal contribuição desse trabalho é a avaliação do desempenho de diferentes classificadores diante das formas alternativas de tratar o dado no contexto SQLIA. Neste sentido, não foi possível determinar um classificador que tenha tido melhor desempenho em todos os casos, evidenciando que não há uma melhor técnica de pré-processamento nesse caso. Essa é uma escolha que deve ser feita em conjunto com o dado e o classificador.

Os classificadores *Random Forest* e *Gradient Boosting Tree* apresentaram as melhores acurácias. Considerando um balanceamento entre acurácia e tempo de processamento, o *Gradient Boosting Tree* aplicado ao conjunto das métricas G-Test e Entropia, calculadas sobre tokenização e transformação realizadas com expressão regular, pode ser considerado o melhor resultado deste trabalho, com acurácia de 98,46%.

Considera-se que o objetivo do trabalho foi alcançado, uma vez que as técnicas de pré-processamento foram implementadas seguindo os trabalhos da literatura e avaliadas conforme acurácia e tempo de processamento de quatro classificadores. Os desempenhos inferiores encontrados neste trabalho em relação aos da literatura podem ter sido influenciados pelo tamanho das bases de dados e pela tabela de expressão regular. No entanto, de forma geral, entende-se que a acurácia de de 98,4% é um resultado satisfatório.

Referências

Aggarwal, C. C. and Zhai, C. (2012). *Mining text data*. Springer Science & Business Media.

Cavnar, W. B. and Trenkle, J. M. (1994). N-gram-based text categorization. In *3rd Annual symp. on document analysis and information retrieval*.

Chen, Z., Guo, M., and Zhou, L. (2018). Research on SQL injection detection technology based on SVM. In *MATEC Web of Conferences*, volume 173, pages 1–5.

Choi, J., Choi, C., Kim, H., and Kim, P. (2011). Efficient malicious code detection using N-gram analysis and SVM. In *International Conference on Network-Based Information Systems, NBiS 2011*, pages 618–621. IEEE.

Fang, Y., Peng, J., Liu, L., and Huang, C. (2018). WOVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM. In *International Conference on Cryptography, Security and Privacy*, pages 170–174, New York, NY, USA. ACM.

Hanmanthu, B., Ram, B. R., and Niranjan, P. (2015). SQL injection attack prevention based on decision tree classification. In *International Conference on Intelligent Systems and Control, ISCO 2015*, page 5. IEEE.

Joshi, A. and Geetha, V. (2014). SQL Injection detection using machine learning. In *International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014*, number 2, pages 1111–1115. IEEE.

Kim, M. Y. and Lee, D. H. (2014). Data-mining based SQL injection attack detection using internal query trees. *Expert Systems with Applications*, 41(11):5416–5430.

Lodeiro-Santiago, M., Caballero-Gil, C., and Caballero-Gil, P. (2017). Collaborative SQL-injections detection system with machine learning. *ACM International Conference Proceeding Series*.

McWhirter, P. R., Kifayat, K., Shi, Q., and Askwith, B. (2018). SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel. *Journal of Information Security and Applications*, 40:199–216.

Mishra, S. (2019). *SQL Injection detection using machine learning*. Dissertação (mestrado), San José State University.

Nayak, R. and Qiu, T. (2005). A data mining application: Analysis of problems occurring during a software project development process. *International Journal of Software Engineering and Knowledge Engineering*, 15(04):647–663.

OWASP, O. W. A. S. P. (2020). OWASP Top 10 Web Application Security Risks. [ONLINE] - https://owasp.org/www-project-top-ten/. Acesso em Maio/2021.

Rankothge, W. H., Randeniya, M., and Samaranayaka, V. (2020). Identification and Mitigation Tool for Sql Injection Attacks (SQLIA). In *International Conference on Industrial and Information Systems (ICIIS)*, pages 591–595. IEEE.

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).