ZONA TEKNIK: JURNAL ILMIAH
Volume x, Number x, 2021 pp. 443-451
P-ISSN 1978-1741, E-ISSN 2721-2912
DOI:http://dx.doi.org/10.37776/zt.vxix.xxx
Open Access: http://ejurnal.univbatam.ac.id/index.php/Teknik

Aplikasi *Machine Learning* Untuk Deteksi Serangan *Code Injection*

Suroto[1*], John Friadi[2], Sultan Septian[3]
[1,2]Prodi Sistem Informasi, Universitas Batam, Batam – Kepri, Indonesia
**Email: suroto@univbatam.ac.id*

A R T I C L E I N F O

Genesis Artikel:
Diterima, 21-2-2022
Direvisi,7-3 2022
Disetujui, 28-32022

Kata Kunci:
Injeksi Kode, Machine Learning, Deep Learning

Keywords:
Code Injection, Machine Learning, Deep Learning

ABSTRACT

Code Injection is one of the top forms of cybersecurity attack in the modern world. To overcome the limitations of conventional signature-based detection techniques, and to complement them, several Machine Learning approaches have been proposed. When analysing this approach, the survey focused primarily on common intrusion detection, which can be further applied to specific vulnerabilities. In addition, between the steps of Machine Learning, there is a data pre-processing phase, which is very important in the data analysis process. This data pre-processing appears to have been the least researched in the context of Network Intrusion detection, i.e. in Code Injection. The aim of this survey is to fill the gap by analysing and classifying existing Machine Learning techniques, which are applied to Code Injection attack detection, with particular attention to Deep Learning . Our analysis reveals that the way input data is pre-processed greatly affects performance and attack detection rates. The proposed pre-processing cycle demonstrates how various Machine Learning -based approaches to detect Code Injection attacks take advantage of different input data pre-processing techniques.

ABSTRAK

Code Injection adalah salah satu bentuk serangan keamanan siber teratas di dunia modern. Untuk mengatasi keterbatasan teknik deteksi berbasis tanda tangan konvensional, dan untuk melengkapinya, beberapa pendekatan pembelajaran mesin telah diusulkan. Saat menganalisis pendekatan ini, survei berfokus terutama pada deteksi intrusi umum, yang dapat diterapkan lebih lanjut untuk kerentanan tertentu. Selain itu, di antara langkah-langkah pembelajaran mesin, ada fase *pre-processing* (pra pemrosesan) data, yang sangat penting dalam proses analisis data. Pra-pemrosesan data ini tampaknya paling sedikit diteliti dalam konteks deteksi Intrusi Jaringan, yaitu dalam *Code Injection*. Tujuan dari survei ini adalah untuk mengisi kesenjangan dengan menganalisis dan mengklasifikasikan teknik *Machine Learning* yang ada, yang diterapkan pada deteksi serangan *Code Injection*, dengan perhatian khusus pada *Deep Learning* . Analisis kami mengungkapkan bahwa cara data input diproses sebelumnya sangat mempengaruhi kinerja dan tingkat deteksi serangan. Siklus prapemrosesan yang diusulkan menunjukkan bagaimana berbagai pendekatan berbasis *Machine Learning* untuk mendeteksi serangan Code *Injection* memanfaatkan teknik prapemrosesan data input yang berbeda.

PENDAHULUAN

Code Injection (injeksi kode) adalah istilah umum untuk jenis serangan yang terdiri dari 'menyuntikkan' kode program yang kemudian dieksekusi oleh aplikasi (OWASP, 2021). Jenis serangan ini mengeksploitasi penanganan *untrusted data* (data yang tidak terpercaya). Serangan ini dimungkinkan karena kurangnya validasi data input/output yang tepat seperti: karakter yang diizinkan (*standard regular expressions classes*), format data dan jumlah data yang diharapkan. Kode disuntikkan dalam bahasa aplikasi yang ditargetkan dan dieksekusi oleh penerjemah sisi server untuk bahasa tersebut, misalnya: PHP, Python, Java, Perl, Ruby, dll. Aplikasi apa pun yang secara langsung mengevaluasi input yang tidak divalidasi, rentan terhadap injeksi kode. Secara umum, aplikasi web adalah target utama bagi penyerang.

Code Injection adalah termasuk serangan sangat populer dan berdampak, yang berada di urutan ketiga dalam daftar *OWASP Top 10 Security Risk* (OWASP, 2021). Bahkan pada tahun 2021, jenis serangan ini menempati urutan teratas. Pendeteksian serangan *Code Injection* secara tradisional dilakukan dengan menggunakan teknik pengenalan berbasis *Sign* atau *Pattern*. Baru-baru ini deteksi serangan dilengkapi dengan penerapan pendekatan *Machine Learning* (pembelajaran mesin) tingkat lanjut. *Machine Learning* merupakan metode algoritma komputer yang memperbaiki secara mekanis melalui pengalaman.(Ananda, 2021). Algoritma *Machine Learning*

membuat model yang mendukung *sample data*, yang disebut sebagai "*training data* (data pelatihan)", sehingga membentuk previsi (penilaian). Berbagai metode *Machine Learning* untuk deteksi intrusi telah bermunculan akhir-akhir ini, diantaranya *Naïve Bayes Classifier, Support Vector Machine* (SVM) dan *Convolutional Neural Network* (CNN). Keuntungan dari teknik tersebut adalah bahwa algoritma serupa, misalnya, *Deep* atau *Convolutional Neural Networks (CNN)*, dapat menemukan aplikasi dalam berbagai skenario deteksi ancaman. Kendala masih ada dalam aplikasi *deep learning*, seperti: melatih ulang *Neural Network* ketika *dataset* diubah, inkonsistensi dengan prediksi *pattern*, menemukan bentuk optimal dari *deep neural network*.
Terdapat beberapa *approach* (pendekatan) dan teknik yang ada dalam mendeteksi serangan *Code Injection*. Ini memunculkan pertanyaan. Pertama, metode *Machine Learning* manakah yang lebih cocok untuk deteksi serangan *Code Injection*. Kedua, manakah pendekatan terbaik untuk meningkatkan kinerja dan akurasi deteksi ?.
Tujuan dari survei ini adalah untuk berkontribusi pada ilmu pengetahuan terkait dengan memaksimalkan kinerja dan efektivitas teknik *Machine Learning* yang digunakan dalam deteksi intrusi, khususnya dalam deteksi serangan *Code Injection*. *Convolutional Neural Networks* (CNN) yang banyak digunakan telah dipilih untuk dianalisis. CNN dipilih untuk eksperimen karena persyaratan komputasi yang rendah. CNN tidak memerlukan output sekuensial untuk implementasi spesifik ini, dan input *dataset* multi-dimensi. CNN adalah jaringan *feedforward* khusus dengan lapisan yang memiliki *set parameter* yang dikurangi, karena untuk pelatihan *filter* terjemahan-invarian dengan *locally-receptive field* terbatas. Selain itu CNN dilengkapi dengan teknik yang memungkinkan langkah-langkah yang optimal secara komputasi. Tujuan survei termasuk revisi metode yang diusulkan dalam publikasi akademik terbaru, identifikasi langkah-langkah umum dalam pendekatan, analisis dan klasifikasi yang telah ada. Perhatian khusus diberikan pada langkah-langkah modular dalam persiapan data mulai dari akuisisi data hingga inisiasi proses pelatihan, serta pemilihan metode *Machine Learning* .
Lingkup survei terbatas untuk meninjau transfer data yang tidak dienkripsi dan fiturnya, yang tidak melibatkan manipulasi dengan *encrypted packet payload,* dan *traffic header*. Isu trafik data dan enkripsi dipisahkan dari penelitian ini karena terkait dengan teknik lain (misal, *cryptanalysis*).

METODE PENELITIAN

Penelitian ini menggunakan pendekatan kualitatif. Pada proses penelitian ini dilakukan beberapa tahapan mulai dari tahap awal yaitu tahap inisiasi, pengembangan model-model, evaluasi model dan tahap pemilihan model. Dalam tahapan inisiasi beberapa kegiatan mulai dari identifikasi permasalahan dan studi literatur untuk menemukan alternatif solusi untuk masalah yang ada. Tahap selanjutnya, yaitu tahap training berbagai model dengan mempersiapkan dataset, tahap evaluasi kandidat model dan pemilihan model algoritma yang sesuai. Di tahapan akhir test model.
Teknik pengumpulan data menggunakan studi literatur / Pustaka. Data-data khususnya *Payload Dataset* serangan *code injection* dikumpulkan dari internet. *Dataset* untuk serangan masing-masing adalah SQL *Payload Dataset* dan XSS *Payload Dataset*. XSS *Payload Dataset* diambil dari repositori GitHub. Tabel 1. menunjukkan asal atau sumber data dari *Dataset* yang digunakan.

Tabel 1. Sumber data dari *Dataset*

Attack Type	*Dataset*	*Data Source*
SQL *Injection*	SQL Payload *Dataset*	https://github.com/SuperCowPowers/datahacking/tree/master/sql*Injection*/data
XSS	XSS Payload *Dataset*	https://github.com/payloadbox/xss-payload-list

Algoritma yang digunakan adalah *Convolutional Neural Network (CNN)*. Pada fase analisis, sebelum pelatihan awal *Neural Network*, *script* mengubah *query* " input " (atau kode yang disuntikkan) menjadi urutan dan *pads* dengan nol untuk mencapai panjang *query* maksimum, membentuk *training dataset* dari file yang disediakan.

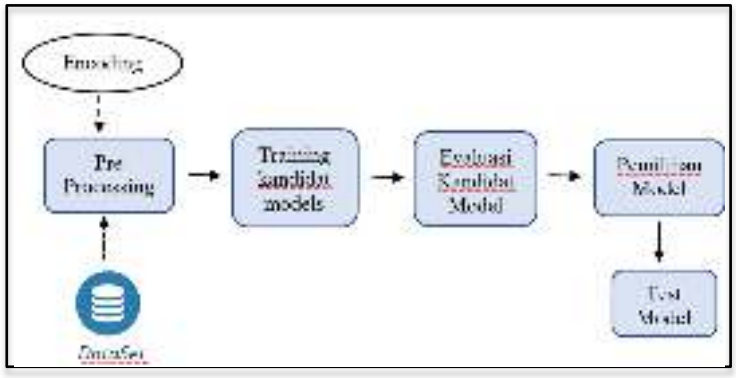
HASIL & PEMBAHASAN

Penulis ini menggunakan *empty space* untuk klasifikasi pra-pemrosesan. Poin penting dari penelitian ini adalah bagaimana membuat string input lebih mudah dibaca untuk model *machine learning*. Penulis menerapkan dua tahap *pre-processing*, yaitu *Removing noise* dan *Symbolize*, untuk mencapai target ini. *Code Injection Detection* menggunakan *Deep Learning* (selanjutnya disingkat CIDDL) bergantung pada *supervised approach*. Selama fase pelatihan, CIDDL diumpankan dengan *dataset* yang terdiri dari entri berlabel (*input quires, injected code payloads*, dll). CIDDL memodifikasi CNN tertentu (misalnya *pre-configured*). CIDDL mengoptimalkan bentuk dan parameter CNN melalui teknik pencarian lokal. Teknik ini dilakukan dengan cara menguji berbagai alternatif model dan memilih set parameter dengan kinerja terbaik. *Deep Neural Network* terlatih yang terpilih kemudian digunakan selama fase online untuk mengambil keputusan atas *query*.

Peran modul *pre-processing* tersebut adalah untuk mempercepat pelatihan dan meningkatkan tingkat deteksi melalui dua strategi yang saling melengkapi:

a) *Removing Noise*: - menyaring informasi "*noisy*" dari input *query* (seperti nama kolom atau tabel), dengan menghapus duplikat dari *dataset*.
b) *Symbolize*: - memperkaya *query* asli dengan label *semantic* (tipe *operator*/simbol).

Konsep dasar tujuan *Neural Network* yang terlatih adalah untuk "memahami" peran simbol atau *operator* tertentu. Percepatan proses tersebut dicapai dengan cara mengubah *query* asli ke dalam *pattern* yang berbeda, serta menambahkan pengetahuan khusus dalam bentuk label *"type"*. Targetnya adalah simbol, ekspresi, *operator* bahasa pemrograman dan sebagainya. Gambar 1, menunjukkan alur kerja dari algoritma training tersebut.



Gambar 1. Alur kerja algoritma *training*

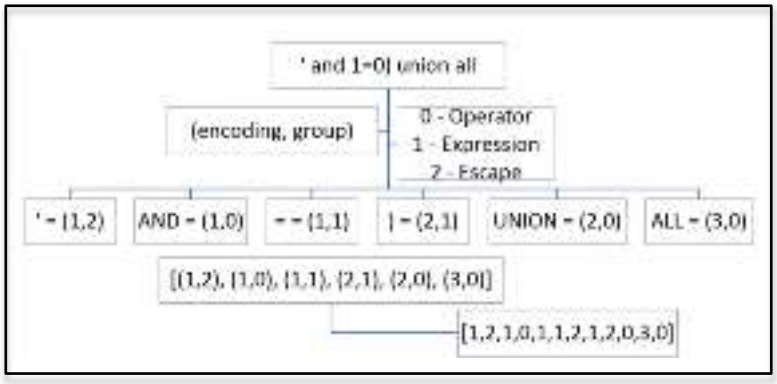
Pada Gambar 1, *Dataset* dan *Encoding* adalah file teks dengan baris kode. *Encoding* digunakan oleh algoritma *pre-processing* untuk mengubah *Dataset* menjadi format yang dapat dibaca oleh CNN menggunakan nilai dari file *Encoding*. Setelah *pre-processing query* dari *Dataset*, dan sebelum menambahkan setiap *array* yang diubah ke panjang tetap, CIDDL membandingkan panjang awal maksimum dengan panjang *query pre-processing* terpanjang dalam *Dataset*. CNN dilatih berulang kali sampai nilai kerugian berhenti meningkat. Untuk *Dataset* kecil, kerugiannya harus di bawah 0,05, atau bahkan 0,02. Batasan ditambahkan ke CIDDL untuk menghentikan pelatihan model dengan satu set konfigurasi, dan pindah ke set konfigurasi lain, karena fakta bahwa dalam banyak kasus nilai kerugian tidak membaik di bawah 0,08.

Jadi secara ringkas, strategi *symbolize* ini menerapkan *encoding* dengan tidak hanya nilai tunggal, tetapi menggunakan pasangan nilai. Satu untuk bagian string mentah dan yang lainnya adalah kode, yang mewakili kategori untuk ramuan string mentah. Nilai masing-masing ditunjukkan dalam table 2 berikut:

Tabel 2. *Encoding* kategori simbol dalam metode yang diusulkan

Kode	Kategori	Contoh
0	*Operators*	AND, UNION, SELECT FROM
1	*Expressions*	=,)
2	*Escape Symbols*	‘

Gambar 2 di bawah ini menyajikan contoh *pre-processing query* sebelum fase *training* atau klasifikasi oleh *Neural Network* . Disini terdapat penghilangan variabel dan nilai, pengklasifikasian dan *encoding*.



Gambar 2. Pembedahan *query* SQL menggunakan metode yang diusulkan

Disini, penulis sajikan beberapa contoh *encoding*.

a) SQL *Injection*::

Raw String: *` and 1=0) union all*
Remove Randomness: *` and =) union all*
Encoding: *(1,2),(1,0),(1,1),(2,1),(2,0),(3,0)*

Raw String: *SELECT column1, columns2, column3 FROM tablename*
Remove Randomness: SELECT , , FROM
Encoding: *(5,0),(3,1),(3,1),(10,0)*

b) XSS

Raw String: <img src=1 href=1 onerror="javascript:" alert(1)">
Remove Randomness: <img src = href = onerror =" javascript : " alert( ) " > </img >
Encoding: (1,0),(4,1),(4,1),(4,1),(5,0), (4,1),(2,1),(10,0),(3,1),(2,1),(3,1),(3,1),(6,1)

Convolutional Neural Network membutuhkan *pattern* dengan panjang tetap untuk input data. Lebar minimal yang cukup dari *Dataset* sama dengan *query pre-processing* terpanjang dan menentukan jumlah *neuron* input.
Pendekatan yang disajikan ini, membutuhkan *query* dalam bentuk *native code* (kode asli). Dalam kasus *query* yang disandikan (mis., *Base64*), data tersebut harus diproses terlebih dahulu dan diubah menjadi kode untuk diteruskan ke sistem. Selain itu, selama *training*, *dataset* harus disaring dan duplikatnya harus dihilangkan karena dapat mempengaruhi presisi dan akurasi sistem.
Percobaan dilakukan dalam beberapa langkah:
1) *Pre-processing dataset* mendeteksi ukuran minimal yang cukup dari *patter* untuk menghindari *padding* yang berlebihan.
2) Identifikasi jumlah optimal *hidden layer* untuk *Neural Network*.
3) Identifikasi ukuran *batch* maksimum dengan jumlah minimal siklus *training*.
4) Evaluasi model dengan menguji *dataset* dan statistik analisis.
Lapisan *output* diatur ke 1 *neuron* untuk mengeluarkan satu variabel, idealnya 1 atau 0, atau persentase diantaranya: berpotensi tidak berbahaya (0,00-0,49) atau berpotensi berbahaya (0,50-1,00). Setelah mencapai akurasi yang baik dan kerugian minimal, tidak ada nilai yang mendekati 0,50, dan nilai tersebut sangat condong ke 0 atau 1. Logika yang sama diterapkan pada *neuron* keluaran kedua untuk atribusi bahasa pemrograman.
Jumlah *neuron* pada lapisan input ditentukan oleh item terbesar dalam *dataset.* Setiap *query* SQL tidak memiliki lebih dari 500 nilai dalam satu *pattern*, sementara setiap baris JavaScript tidak memiliki lebih dari 3000 nilai. Bentuk segitiga *Neural Network* memastikan bahwa *pattern* 500-3000 angka diproses hanya memiliki satu nilai di lapisan *output*. Setiap lapisan berikutnya secara bertahap memiliki lebih sedikit *neuron* daripada yang sebelumnya. Misalnya, lapisan input diatur ke 500 *neuron*, *hidden layer* (lapisan tersembunyi) pertama - 600 *neuron*, dan yang kedua (jika ada) hingga 100 *neuron*. *Neuron* keluaran tetap 1 untuk semua percobaan.
Setelah *pre-processing* dan penghapusan duplikat, *dataset* dibagi 80%/20%, dimana 80% dari *dataset* (baik berbahaya dan tidak berbahaya) digunakan untuk melatih *Deep Neural Network*, dan 20% dari *dataset* digunakan untuk mengevaluasi akurasi, presisi, dan mengingat *dataset*. Akurasi rata-rata antara 92% dan 94%. Atau, *dataset* dibagi 70/30% dan 50/50% untuk evaluasi.

Dataset yang tidak seimbang dengan duplikat menimbulkan bias ke dalam model *neural*, dan model mendeteksi semua *query* sebagai berbahaya atau tidak berbahaya. Sebelum pengujian awal, kami menjalankan CIDDL tanpa fungsi *pre-processing* (lihat table 4.), cukup mengkodekan setiap simbol *query* ke dalam *charcode*. Jenis *encoding* ini memiliki batasan terhadap serangan yang mengubah urutan simbol dalam injeksi *query*. Parameter kerugian tidak pernah turun di bawah 0,30 dalam kondisi yang sama.

Table 4. Unjukkerja *Neural Network* pada *dataset* untuk deteksi SQL *Injection* tanpa *pre-processing*.

Dataset Split	*Hidden Layers*	*Batch Size*	*Accuracy*	*Precision*	*Recall*
80% | 20%	2	2000	74.0%	84.2%	82.9%
70% | 30%	2	2000	71.7%	85.3%	81.7%
50% | 50%	2	2000	57.8%	80.0%	54.9%
80% | 20%	1	1000	73.4%	87.6%	80.7%
50% | 50%	1	1000	63.9%	78.2%	75.6%

Seperti yang ditunjukkan oleh eksperimen, dengan nilai kerugian 0,10-0,07 dimungkinkan untuk mencapai akurasi prediksi serangan yang relatif tinggi dengan *dataset* yang kecil. Bentuk *Neural Network* dalam contoh khusus ini adalah sebagai berikut: 500 *neuron* pada lapisan input, 600 *neuron* pada lapisan tersembunyi pertama, 400 *neuron* pada lapisan tersembunyi kedua, dan 1 *neuron* pada lapisan keluaran.
Tanpa *pre-processing*, hasil deteksi tidak optimal untuk sistem deteksi ancaman. Implementasi *pre-processing* membantu *Neural Network* dengan pengenalan *pattern*, memberikan pengetahuan tambahan tentang *query* dan bahasa, sehingga meningkatkan tingkat deteksi untuk bahasa yang dipilih.

Table 5. Unjukkerja *Neural Network* pada *dataset* untuk deteksi SQL *Injection* dengan *pre-processing*

Dataset Split	*Hidden Layers*	*Batch Size*	*Accuracy*	*Precision*	*Recall*
80% | 20%	2	2000	95.7%	99.0%	91.2%
70% | 30%	2	2000	90.1%	90.1%	90.1%
50% | 50%	2	2000	85.0%	93.3%	89.0%
80% | 20%	1	1000	94.0%	97.8%	96.2%
50% | 50%	1	1000	93.8%	96.5%	95.1%

Pada tabel 5 dan table 6, pemisahan *dataset* mengurangi jumlah sampel *training* dan meningkatkan jumlah sampel pengujian. Kompleksitas meningkat untuk model *neural* dan akurasi *output* berkurang, sebab *dataset training* berbeda dari *dataset* pengujian,. Namun, masalah ini sebagian dapat diselesaikan dengan meningkatkan jumlah siklus *training*. Misalnya, *training* model *split* 80/20 dapat memakan waktu 20 *epoch*, yang *training* model *split* 50/50 dapat memakan waktu hingga 50 *epoch*.

Table 6. Unjukkerja *Neural Network* pada *dataset* untuk deteksi serangan XSS dengan *pre-processing*

Dataset Split	Hidden Layers	Batch Size	Accuracy	Precision	Recall
80% | 20%	2	2000	95.7%	99.0%	91.2%
70% | 30%	2	2000	90.1%	90.1%	90.1%
50% | 50%	2	2000	85.0%	93.3%	89.0%
80% | 20%	1	1000	94.0%	97.8%	96.2%
50% | 50%	1	1000	93.8%	96.5%	95.1%

Dataset SQL yang tidak seimbang berisi 13000 entri berbahaya dan 1020 entri yang sah dan menghasilkan akurasi deteksi 92,2-94,0%. Hasil tersebut dicapai dengan 20-22 *epoch training*, tanpa *overtraining* sistem. *Dataset* yang lebih besar akan membutuhkan lebih banyak waktu untuk dilatih, dan berpotensi lebih sedikit, karena akan memiliki lebih banyak sampel.

KESIMPULAN

Makalah ini telah mengusulkan CIDDL, sebuah sistem yang dirancang untuk mendeteksi serangan injeksi kode menggunakan *Convolutional (Deep) Neural Network*. CIDDL memperkenalkan teknik *pre-processing*. Daripada memproses data injeksi kode mentah, CIDDL mengubah data asli menjadi *encoded pattern* (pola yang dikodekan). Di satu sisi, *encoding* dapat menghilangkan keacakan dalam data asli. Di sisi lain, CIDDL mengkodekan setiap simbol, perintah, atau instruksi yang awalnya bersifat atomik ke dalam pasangan <code,type>. Ini memungkinkan tidak hanya untuk menyimpan informasi tentang simbol asli, tetapi juga untuk menambahkan label semantik sederhana yang membantu *Neural Network* untuk "memahami" peran simbol atau *operator* tertentu itu sendiri, sehingga secara signifikan mengurangi kebutuhan pelatihan.

Hasil numerik telah dihasilkan pada dua *dataset* nyata yang mencakup serangan SQL dan XSS. Hasilnya menunjukkan bahwa CIDDL memberikan kinerja deteksi deteksi yang sangat baik, akurasi hingga 93%, presisi 98%, dan nilai *recall* 93%.

REFERENSI

Athailah, Friadi, J. (2017). Sistem Informasi Manajemen Aset Menggunakan Pendekatan MVC dengan Frmework CodeIginiter di PT. H-Tech Oilfield Equipment. *Zona Komputer*, *7*.
Afam O. (2018). How Does Code Injection Work?. Diakses pada 6 Januari 2022 dari https://www.maketecheasier.com/how-does-code-*Injection*-work/
Augusto V., Pradip B., Alper B. (2017). *Rugged Embedded Systems: Computing in Harsh Environments*. Cambridge MA 02139, United States: Elsevier Inc.
Borislav K. (2021). A Guide to Command Injection – Examples, Testing, Prevention. Diakses pada 6 Januari 2022

dari https://crashtest-security.com/*Command-Injection*/
Cai Ruichu, Xu Boyan, Zhang Zhenjie, Yang Xiaoyan, Li Zijian, Liang Zhihao. (2018). An encoder-decoder framework translating natural language to database queries. *IJCAI International Joint Conference on artificial intelligence,* July. 3977–83.
Dong Ying, Zhang Yuqing, Ma Hua, Wu Qianru, Liu Qixu, Wang Kai, Wang Wenjie. (2018). An adaptive system for detecting malicious queries in web attacks. *Journal of Science China Information Sciences*, 61(3). https://doi.org/10.1007/s11432-017-9288-4.
E. Edalat, B. Sadeghiyan, and F. Ghassemi. (2018). ConsiDroid: A concolic-based tool for detecting SQL Injection vulnerability in Android apps. 2018(Nov), 1-10. Diakses dari: https://arxiv.org/abs/1811.10448
Fang Yong, Yang Li, Liu Liang, Huang Cheng. (2018). DeepXSS: Cross Site Scripting Detection Based on Deep Learning. *Proceeding of the 2018 International Association for computing machinery (ACM),* March, 47–51. https://doi.org/10.1145/3194452.3194469.
Friadi, J. (2021). *Design of Religious Tourism Information System in the Batam City Based on Android Smartphone Corresponding Author :* 17–22.
Friadi, J., Agestira, D., Rumayar, M. A., Dewiwin, N., & Friadi, J. (2022). Sosialisasi dan Penyuluhan Strategi Pemasaran Digital Pada UMKM Baby Smart Bubur Bayi Berbasis E-Commerce. *Jurnal Pengabdian Barelang*, *4*(1), 71–77.
Friadi, J., & Gulo, J. R. (2020). *Pengembangan Sistem Informasi Monitoring Prakrind Dengan Model Rapid Application Development*. 222–229.
Krismadinata, Verawardina, U., Jalinus, N., Rizal, F., Sukardi, Sudira, P., Ramadhani, D., Lubis, A. L., Friadi, J., Arifin, A. S. R., & Novaliendry, D. (2020). Blended learning as instructional model in vocational education: Literature review. *Universal Journal of Educational Research*, *8*(11B). https://doi.org/10.13189/ujer.2020.082214
OWASP Top 10:2021. Introduction: Welcome to the OWASP Top 10 - 2021. Diakses pada 5 Januari 2022 dari https://owasp.org/Top10/
Pi, R., Notifications, W., Kurniawan, D. E., Iqbal, M., Friadi, J., & Borman, R. I. (2019). *Smart Monitoring Temperature and Humidity of the Room Server Using Smart Monitoring Temperature and Humidity of the Room Server Using Raspberry Pi and Whatsapp Notifications*. https://doi.org/10.1088/1742-6596/1351/1/012006
Portswigger. (2021). OS command injection. *Web Security Academy*. Diakses pada 5 Januari 2022 dari https://portswigger.net/web-security/os-*Command-Injection*
R. Cai, B. Xu, Z. Zhang, X. Yang, Z. Li, and Z. Liang.(2018). An encoder-decoder framework translating natural language to database queries. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence.* 2018(Nov), 3977–3983. https://doi.org/10.24963/ijcai.2018/553
S.S. Anandha Krishnan; Adhil N Sabu; Priya P Sajan; A.L. Sreedeep; SQL *Injection* Detection Using Machine Learning . *Journal Geintec* .11(3)
Sudip S. (2021). Code Injection – Examples and Prevention. Diakses pada 5 Januari 2022 dari https://crashtest-security.com/code-*Injection*/
Weilin Z., Rezos. (2021). Code Injection. *OWASP Community Pages*. Diakses pada 5 Januari 2022 dari https://owasp.org/www-community/attacks/Code_Injection
Yan Ruibo, Xiao Xi, Hu Guangwu, Peng Sancheng, Jiang Yong. (2018). New deep learning method to detect code Injection attacks on hybrid applications. *Journal System Software*. 137(March),67–77. https://doi.org/10.1016/j.jss.2017.11.001.
Yingbo Li, Bin Zhang. (2019). Detection of SQL Injection Attacks Based on Improved TFIDF Algorithm. *Journal of Physics: Conference Series*, 1395(1), 100-115. http://dx.doi.org/10.1088/1742-6596/1395/1/012013
Zbigniew B. (2019). What is Code *Injection* and How to Avoid It. Diakses pada 5 Januari 2022 dari https://www.netsparker.com/blog/web-security/code-Injection/