Web Application Attack Detection using Deep Learning

Firdausi Aditya Darmawan, 18217001, Sistem Teknologi Informasi
Makalah Keamanan Informasi

Abstraksi—Aplikasi web menjadi begitu popular pada era internet hari ini. Hal ini membuat aplikasi web merupakan target yang paling sering terjadinya *cyber-attack*. Karena, aplikasi web dapat sangat mudah diakses melalu jaringan dan sangat rentan sekali. Serangan terhadap web dapat menyebabkan kerusakan dan kehilangan data serta infromasi penting yang disimpan. Sistem monitoring deteksi aplikasi web dan peringatan saat sebuah serangan terdeteksi. Oleh karena itu, perlu adanya perlindungan yang harus dibuat sekarang untuk melindungi atau mendeteksi saat terjadi serangan terhadap web yang lebih baik dan kompleks. Dalam mendeteksi serangan web, salah satunya dengan menggunakaan *deep learning*. Hal yang dilakukan adalah melakukan *training* terhadap *request* yang diterima oleh web. Dalam melakukan training, data akan dikelompokkan dan diberi label menjadi request yang normal dan anomali. Hasil training akan membentuk model untuk mengetahui pola request normal dan anomali terhadap web. Model yang telah dibuat tersebut akan dilakukan proses *testing* untuk menguji keakuratan dari model yang telah dibuat.

Kata kunci—aplikasi web*, deep learning, cyber-attack, request*

1. PENDAHULUAN

Pada era industry 4.0 ini, merupakan era digital menggunakan berbagai teknologi maju yang telah dikembangkan. Teknologi ini telah diterapkan dalam semua aspek kehidupan masyarakat dengan dilakukannya transformasi digital yang banyak diperbincangkan sekarang. Salah satunya, adalah aplikasi web memiliki peran yang sangat penting pada kehidupan sehari-hari sekarang. Transformasi ini dimulai ketika orang maupun kelompok orang memindahkan data pribadi dan informasi penting ke *cloud*. Data yang disimpan secara digital tersebut merupakan target yang menjadi serangan. Aplikasi web menjadi sektor utama yang menjadi target penyerang dan sampai sekarang masih terjadi terus menerus. Terdapat berbagai macam laporan terkait *cyber-attack* yang terorganisir menyerang situs web setiap harinya. Sehingga, perlindungan aplikasi web merupakan bagian yang sangat vital.

Aplikasi web merupakan target penyerang secara terus-menerus. Di antara semua jenis serangan dan kerentanan, kerentanan terkait *request* HTTP yang dieksploitasi dengan mengirim *request* yang tidak wajar merupakan yang terbanyak dilakukan. Menurut [8], *SQL injection, DDoS, cross site scripting* (XSS) merupkan tingkat resiko keamanan yang paling serius. Serangan tersebut dapat menonaktifkan layanan web, resiko keamanan perusahaan, kehilangan data informasi bagi penyedia dan pengguna layanan, bahkan sampai kerugian finansial yang sangat besar. Meskipun, pengembang telah berusaha untuk mencegah terjadinya serangan dengan menggunakan *firewall* dan *intrusion detection system*. Serangan aplikasi web tetap masih menjadi ancaman utama.

Sebagai contoh, berdasarkan data statistik yang diperoleh terkait kerentanan dan ancaman pada aplikasi web. Menurut [2], *Unauthorized access* mungkin terjadi di 39% aplikasi web, *broken authentication* terjadi pada 45% aplikasi web. Pelanggaran informasi penting merupakan ancaman yang cukup mendesak untuk keamanan aplikasi web. Hampir setengah dari pelanggaran (47%) merupakan personal data yang berisiko. Pengguna kredensial juga cukup menonjol (31%). Dari statistik menunjukkan bahwa informasi pribadi merupakan target utama penyerang ketika menargetkan sebuah organisasi atau perusahaan.

Pada umumnya, terdapat dua pendekatan untuk mendeteksi serangan tersebut. Pertama, menggunakan metode *signature-based* yang melihat pola serangan secara spesifik pada *request* HTTP. Kedua, menggunakan metode *anomaly-based* yang membentuk profil pengenal *request* normal. Sehingga, dapat membedakan *request* anomali dengan *request* normal. Penggunaan metode *signature-based* memiliki keakuratan yang lebih tinggi dengan kesalahan notifikasi peringatan yang sedikit daripada dengan *anomaly-based*. Sebagai contoh, *Web Application Firewall* (WAF) dan *Core Rule Set* (CRS) yang dapat mendeteksi *SQL Injection*, *Cross Site Scripting*, dan *HTTP Protocol Violation*.

Metode *signature-based* efektif dalam mendeteksi serangan, tetapi metode ini masih memiliki beberapa masalah. Pertama, metode ini hanya bagus pada dataset yang telah ditetapkan. Sehingga, tidak mampu mendeteksi serangan diluar dataset tersebut. Untuk itu, perlu banyak jumlah dataset yang telah diberi label sebagai anomali agar metode ini bagus. Namun, kesulitan dan mahalnya untuk mendapatkan dataset tersebut. Selain itu, dataset yang dibuat juga tidak seimbang. Karena, dataset yang berlabel normal sangat jauh lebih banyak daripada yang berlabel anomali. Kedua, pola serangan atau request yang sangat banyak membutuhkan sumber daya

komputasi yang besar. Sebagai contoh, akan sangat sulit untuk tetap menjaga agar *signature* dataset tersebut selalu update untuk kerentanan yang terjadi setiap hari.

Untuk mengatasi masalah yang ada menggunakan metode *signature-based*, digunakan metode *anomaly*-based yang dapat memberikan dukungan dalam mendeteksi serangan baru. Selain itu, metode *anomaly-based* dapat dilatih untuk mendeteksi serangan yang dilakukan terhadap aplikasi web. Dengan menerapkan metode ini, dapat mendeteksi serangan aplikasi web secara otonom, *real-time*, dan beradaptasi secara efektif, efisien, terukur, dan aman.

Pada makalah ini, akan dilakukan metode *anomaly-based* dengan menggunakan *deep learning* yaitu, *Recurrent Neural Network* (RNN) dengan *Long-Short Term Memory* (LSTM) atau *Gate Recurrent Unit* (GRU). Metode *anomaly-based* ini akan mengambil input dengan menggunakan *Uniform Resources Locators* (URL) dalam *request* pada HTTP. URL yang diperoleh akan diberi token untuk mengambil struktur penting pada URL yang digunakan sebagai parameter dan informasi yang terdapat pada URL tersebut. RNN yang telah dilatih untuk mempelajari pola *request* normal akan menerima URL yang diberi token. Dan dengan *neural network* terlatih akan meberikan keputusan *request* yang diberikan merupakan kelompok anomali berdasarkan pengelompokkan yang telah dibuat RNN sebelumnya. Model ini akan melakukan *self-learning* dalam mempelajari pola *request* normal pada aplikasi web.

Pada bagian makalah ini disusun sebagai berikut: Bagian 2 akan merangkum secara singkat tentang jurnal terbaru mengenai deteksi anomali aplikasi web. Bagian 3 akan menjelaskan desain dan arsitektur model yang akan digunakan untuk menganalisis data. Bagian 4 akan menjelaskan percobaan yang yang akan dilakukan dengan teknik yang telah didesain sebelumnya. Bagian 5 akan menjelaskan kesimpulan. Bagian 6 berisi referensi yang digunakan dalam makalah ini.

2. STUDI TERKAIT

K. Christopher. [11] membuat model deteksi *anomaly-based* dengan menggunakan atribut struktur *request* HTTP yang berbeda-beda. Sebagai contoh, dengan mengelompokkan berdasarkan panjang atribut, distribusi karakter atribut, inferensi structural, *token finder*, ada tidaknya atribut, dan urutan dari atribut. Setiap kelompok dari atribut tersebut akan dilakukan *testing* menggunakan model yang telah dibuat untuk menghasilkan nilai probabilitas kemungkinan terjadinya anomali. Untuk menetapkan sebuah *query* merupakan anomali *request* dengan melihat probabilitas yang dihasilkan melewati ambang batas normal yang telah ditetapkan.

L. Duc. [4] membuat model menggunakan *unsupervised learning system* untuk menganalisis dan mendeteksi adanya tindakan jahat berdasarkan perilaku. Dengan menggunakan *Self Organizing Map* (SOM) untuk melakukan *training* pada dataset *network traffic* dan *web request*. Pada akhirnya, akan dilakukan tiga tahapan, yaitu desain, implementasi, dan evaluasi terhadap *unsupervised* SOM *training*.

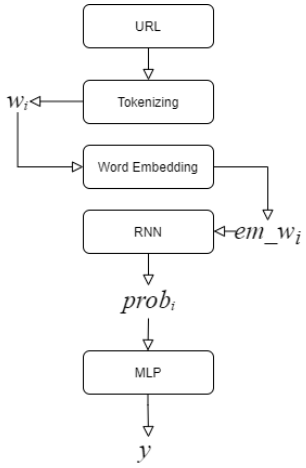
3. DESAIN DAN ARSITEKTUR MODEL

Dalam mendesain model, dengan dasar penggunaan GET *request* untuk mengidentifikasi terjadinya anomali pada URL. Sebuah URL yang memiliki parameter *query* dengan *syntax* umum

"http://" localhost [":" port] [path ["?" query]]

Dengan penjelasan bahwa path merupakan file atau folder yang digunakan untuk mengakses file dalam server. Sedangkan, untuk *query* merupakan sebuah parameter dalam mengakses file. Dengan melihat parameter dan value yang diberikan pada URL dapat mengetahui informasi dalam deteksi anomali. Sebagai contoh, email=xxx@gmail.com merupakan hal yang normal, email=abc merupakan sesuatu yang anomali karena *value* yang diberikan tidak sesuai dengan standar *value* dari sebuah email.

Berdasarkan desain tersebut, dibuat arsitektur model dengan melakukan *training* RNN menggunakan dataset token URL. Dataset URL dinotasikan $U = \{u_1, u_2, u_3 \ldots\}$ dengan u_i merupakan urutan URL ke-i. Dan $y_i \in \{0, 1\}$ dengan indikasi anomali bernilai $y_i = 1$. URL yang akan digunakan harus diberi token terlebih dahulu untuk mendapatkan struktur penting dari URL yang akan di *training*, dinotasikan $\overrightarrow{W} := [w_1, w_2, w_3 \ldots]$ dengan w_i merupakan urutan token ke-i. Untuk merepresentasikan token menjadi vector kontinu, setiap token dipetakan menggunakan *word embedding*. Selain itu, RNN juga dilakukan *training* menggunakan normal *request* untuk mengetahui pola *request* normal. Kemudian, RNN dapat memberikan probabilitas keberhasilan dalam melakukan prediksi terhadap token w_i yang akan muncul. Langkah terakhir, dengan melatih $(\overrightarrow{prob_i}, y_i)$ untuk mengetahui URL yang muncul normal atau anomali. Sehingga, dapat dilakukan deteksi terhadap *request* pada aplikasi web. [7] Berikut diagram dari proses tersebut.

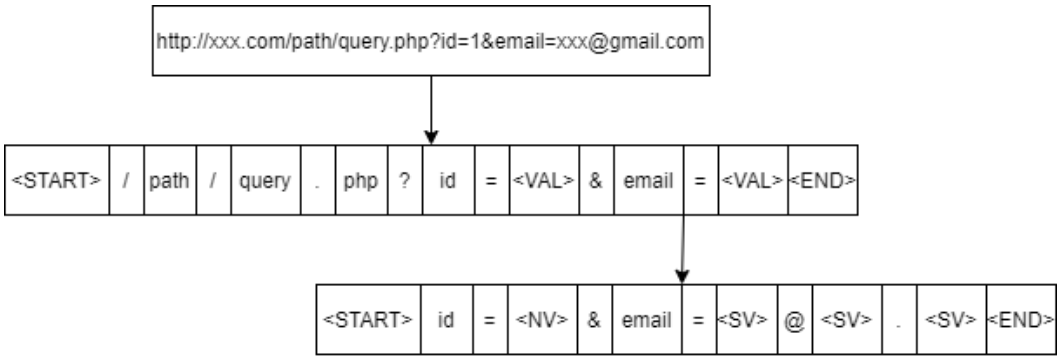


Gambar 1. Diagram model arsitektur

3.1 *TOKENIZING*

Request anomali dapat dibagi ke dalam dua tipe. Untuk tipe pertama, adalah *abnormal query* dengan kesalahan format data yang diberikan. Tipe kedua, adalah anomaly URL yang terdapat pada path. URL *tokenizing* ini digunakan untuk mendapatkan informasi penting yang dibutuhkan pada *query* dan path. Berikut prosedur dari *tokenizing*.

- Mengubah URL menjadi *lowercase*
- Mengubah *query value* dengan <VAL> token
- Membagi *query parameter* berdasarkan *non-word*, *value* numeric dengan <NV>, dan *value* string dengan <SV>
- Menambah <START> dan <END> pada awal dan akhir URL
- Membuat kosakata *training* setidaknya x% token
- Token diluar dari kosakata diberi <UNK> token.



Gambar 2. URL *tokenizing*

3.2 *WORD EMBEDDING*

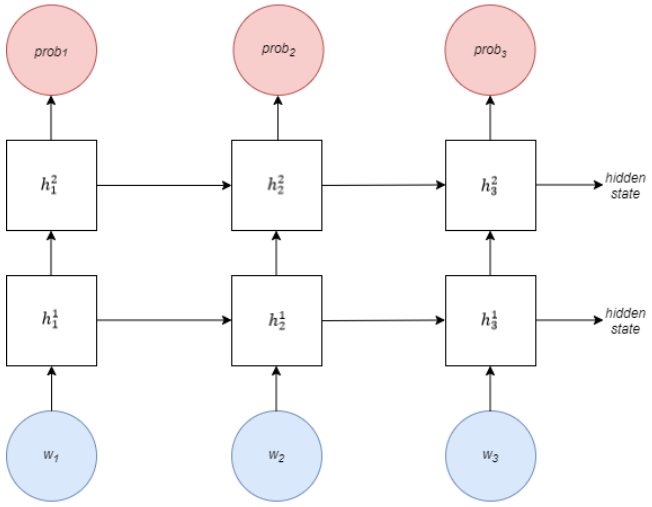
Word embedding merupakan salah satu konsep yang ada di *neural network* yang memetakan kata menjadi vector numerik. *Word embedding* merepresentasikan kata dalam vector secara kontinu dengan dimensi yang lebih rendah. Pada model ini, dapat diibiratkan seperti sebuah token yang akan muncul. Lookup tabel token dapat direkontruksi menjadi baris yang direprsentasikan menggunakan *embedding vector* yang dapat ditulis dalam rumus.

$$\overrightarrow{em\_w} = \vec{w}\overrightarrow{M_{look\_up}}$$

Dengan $\vec{w} \in R^V$ merupakan *one-hot vector* dari token w, $M_{look\_up} \in R^{V*D}$ adalah lookup tabel. V adalah ukuran kosakata dan D ukuran vector.

3.3 *RECURRENT NEURAL NETWORK*

RNN merupakan sebuah *artificial neural network* yang dapat diterapkan pada *sequence*. RNN dapat melakukan pemrosesan pada setiap elemen pada *sequence* tersebut, lalu hasil output tersebut dengan berdasarkan komputasi sebelumnya. RNN dapat disebut menyimpan informasi yang dihasilkan dari proses sebelumnya. Pada model ini, RNN dilakukan untuk mendapatkan probabilitas dari token sebuah URL dengan dilatih menggunakan pola *request* normal. Input dari RNN dinotasikan $\overrightarrow{x_t} := [\overrightarrow{w_1}, \overrightarrow{w_2}, \overrightarrow{w_3} \ldots]$. Dinotasikan h_t^l merupakan sebuah *hidden state* dengan layer *l* pada *timestep* ke-t. Nilai dari h_t^l dapat dihitung atau dikalkulasikan dengan *hidden state* dan *timestep* sebelumnya. Berikut diagram struktur dari RNN.



Gambar 3. Diagram struktur layer RNN

$\overrightarrow{h_t^1}$ =$rnn\left(\overrightarrow{w_t} + \overrightarrow{h_{t-1}^1}\right)$

$\overrightarrow{h_t^l}$ =$rnn\left(\overrightarrow{h_t^{l-1}} + \overrightarrow{h_{t-1}^l}\right)$, dengan $1 < l \leq L$

Variabel L merupakan jumlah layer yang ada pada RNN. Dalam RNN biasanya fungsi bersifat non-linier seperti tanh. Sehingga, persamaan diatas dapat diganti menjadi

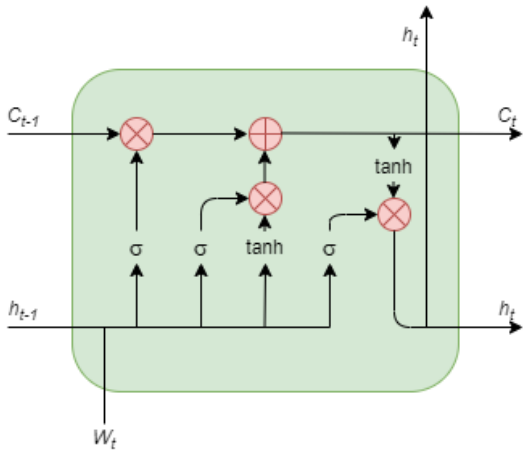
$rnn\left(\overrightarrow{h_t^{l-1}} + \overrightarrow{h_{t-1}^l}\right) := \tanh(\overrightarrow{h_t^{l-1}}W + \overrightarrow{h_{t-1}^l}U)$

Output yang diberikan RNN pada setiap *timestep* adalah nilai yang merepresentasikan probabilitas dari token URL.

3.4 *LONG-SHORT TERN MEMORY*

LSTM merupakan salah satu model pemrosesan lain dari RNN. Pada LSTM, memiliki *hidden state* yang sama seperti RNN sebelumnya yang bernama LSTM *cell*. Namun, LSTM didesain untuk mengatasi terjadinya *vanishing* gradient menggunakan gate.

Pada LSTM *cell* mendapatkan input dari layer sebelumnya $\overrightarrow{h_t^{l-1}}$, *timestep* sebelumnya $\overrightarrow{h_{t-1}^l}$, dan *cell state* sebelumnya $\overrightarrow{C_{t-1}^l}$. Langah pertama, adalah menentukan informasi yang akan dihilangkan pada $\overrightarrow{C_{t-1}^l}$dengan menggunakan *forget gate* $\overrightarrow{f_t^l}$. Langkah selanjutnya, dengan melakukan komputasi pada $\overrightarrow{C_t^l}$ untuk menentukan informasi baru yang akan digunakan. Kemudian, akan dibagi menjadi 2 bagian yaitu, menggunakan *input gate* $\overrightarrow{\iota_t^l}$ untuk menentukan informasi yang akan diperbarui dan vector dari $\overrightarrow{\widetilde{C_t^l}}$ menggunakan tanh. Saatnya, untuk memperbarui $\overrightarrow{C_{t-1}^l}$ menjadi $\overrightarrow{C_t^l}$. Dengan mengalikan $\overrightarrow{C_{t-1}^l}$ dengan $\overrightarrow{f_t^l}$ untuk menghilangkan informasi yang tidak penting dan mengalikan $\overrightarrow{C_t^l}$ dengan $\overrightarrow{\iota_t^l}$ untuk memperbarui informasi. Akhirnya, *output gate* o_t^l dikomputasi untuk mendapatkan output yang diinginkan. Berikut diagram struktur dalam LSTM *cell*.



Gambar 4. Diagram struktur LSTM *cell*

$$\overrightarrow{f_t^l} = \sigma(\overrightarrow{h_{t-1}^l} W_t^l + \overrightarrow{h_{t-1}^l} U_f^l + \overrightarrow{b_c^l})$$

$$\overrightarrow{\imath_t^l} = \sigma(\overrightarrow{h_t^{l-1}} W_i^l + \overrightarrow{h_{t-1}^l} U_i^l + \overrightarrow{b_i^l})$$

$$\overrightarrow{\tilde{C}_t} = tanh(\overrightarrow{h_t^{l-1}} W_C^l + \overrightarrow{h_{t-1}^l} U_C^l + \overrightarrow{b_C^l})$$

$$\overrightarrow{C_t^l} = \overrightarrow{f_t^l} * \overrightarrow{C_{t-1}^l} + \overrightarrow{\imath_t^l} * \overrightarrow{\tilde{C}_t^l}$$

$$\overrightarrow{o_t^l} = \sigma(\overrightarrow{h_t^{l-1}} W_o^l + \overrightarrow{h_{t-1}^l} U_o^l + \overrightarrow{b_o^l})$$

$$\overrightarrow{h_t^l} = \overrightarrow{o_t^l} * \overrightarrow{C_t^l}$$

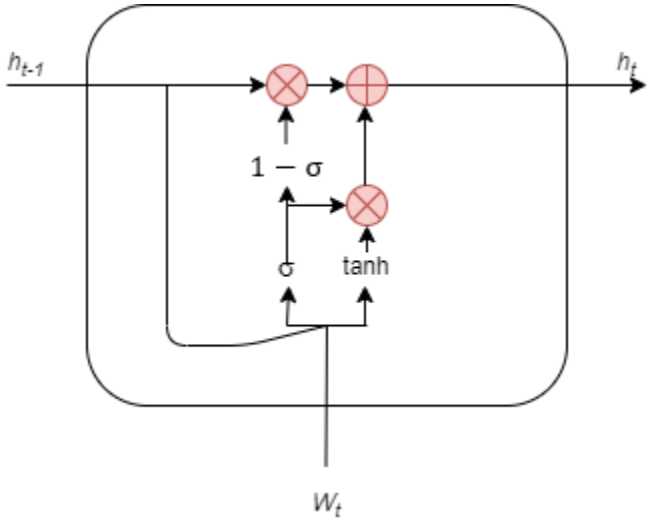
Dengan fungsi σ didefinisikan sebagai $\sigma(x) = \frac{1}{1+e^{-x}}$ dan fungsi tanh didefinisikan sebagai $\tanh(x) = \frac{1- e^{-2x}}{1+ e^{-2x}}$.

3.5 *GATED RECURRENT UNIT*

GRU merupakan pemrosesan yang hampir sama dengan LSTM. Namun, GRU memiliki struktur yang lebih sederhana dengan akurasi yang setara dengan LSTM dan mampu mengatasi *vanishing gradient*.

Pada GRU, menggunakan *update gate* $\overrightarrow{z_t^l}$ dan $\overrightarrow{r_t^l}$ yang memiliki fungsi yang sama dengan *input* dan *forget gate* pada LSTM untuk memberikan informasi yang akan digunakan. GRU *cell* mendapatkan input dari layer sebelumnya $\overrightarrow{h_t^{l-1}}$ dan *timestep* sebelumnya $\overrightarrow{h_{t-1}^l}$. Pertama, menentukan *reset gate* $\overrightarrow{r_t^l}$ yang akan digunakan dalam menghintung $\overrightarrow{\tilde{h}_t^l}$. Kemudian, membentuk $\overrightarrow{\tilde{h}_t^l}$ dengan menggabungkan $\overrightarrow{h_{t-1}^l}$ dengan $\overrightarrow{r_t^l}$ untuk

menghilangkan informasi sebelumnya yang tidak perlu. Kemudian, menggunakan *update gate* $\overrightarrow{u_t^l}$ untuk menentukan informasi yang akan digunakan pada $\overrightarrow{h_t^l}$. Berikut diagram struktur dari GRU *cell*.



Gambar 5. Diagram struktur GRU *cell*

$$\overrightarrow{r_t^l} = \sigma(\overrightarrow{h_t^{l-1}} W_r^l + \overrightarrow{h_{t-1}^l} U_r^l + \overrightarrow{b_r^l})$$

$$\overrightarrow{h_t^l} = tanh(\overrightarrow{h_t^{l-1}} W_h^l + \left(\overrightarrow{r_t^l} * \overrightarrow{h_{t-1}^l}\right) U_h^l)$$

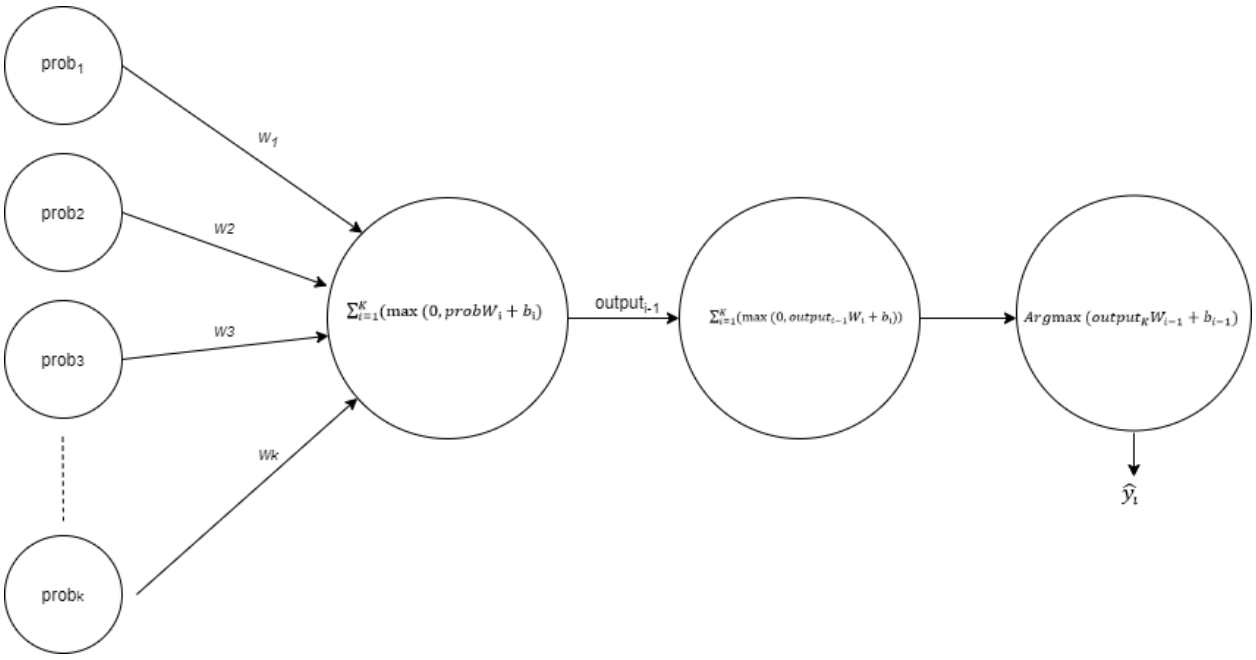
$$\overrightarrow{z_t^l} = \sigma(\overrightarrow{h_t^{l-1}} W_z^l + \overrightarrow{h_{t-1}^l} U_z^l + \overrightarrow{b_z^l})$$

$$\overrightarrow{h_t^l} = \overrightarrow{z_t^l} * \overrightarrow{h_t^l} + \left(1 - \overrightarrow{z_t^l}\right) * \overrightarrow{h_{t-1}^l}$$

3.6 *MULTILAYER PERCEPTRON*

Multilayer Perceptron (MLP) merupakan sebuah *feedforward* dari *artificial neural network*. Pada model ini, MLP digunakan untuk membedakan probabilitas dari *request* normal dan *request* anomali pada URL. URL yang digunakan merupakan output yang dihasilkan oleh RNN pada model.

Fungsi $\hat{y}_i$ = f ($\overrightarrow{prob_i}$), dengan $\hat{y}_i \in$ {normal, anomali}. Berikut merupakan diagram proses dari layer pada MLP tersebut.



Gambar 6. Diagram proses MLP

Dengan W_i merupakan *weight matrix* yang menghubungkan antara input layer dengan hidden layer dan b_i yang merupakan bias *vector*. Setiap input pada layer akan dilakukan operasi secara linier, kemudian hasilnya akan dikomputasi menggunakan fungsi aktivitas yaitu, fungsi maksimum pada diagram. Dan fungsi argmax merupakan prediksi untuk mendapatkan nilai $\hat{y}_i$.

4. EKSPERIMEN

Model yang telah dibuat akan dites terhadap dataset. Akan dilakukan tes menggunakan dua dataset sebagai perbandingan performa model. Untuk mendapatkan performa dari model akan dilakukan pembagian dataset menjadi *training set* dan *test set.*

4.1 DATASET

Dataset pertama yang akan digunakan adalah HTTP dataset CSIC [9]. Dataset tersebut berisi 36.000 *request* normal dan 25.000 *request* anomali termasuk serangan seperti *SQL Injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS* dll. Pada dataset, hanya akan diambil URL GET *request* saja yang akan menjadi dataset. Berikut table dari dataset tersebut.

Tipe	*Training Set*	*Test Set*	Total
Normal	7464	1866	9330
Static Attack	941	246	1187
Dynamic Attack	2651	662	3313
Illegal Request	2900	715	3615

Tabel 1. HTTP dataset CISC

Dataset kedua yang akan digunakan menggunakan WAF. Dataset berisi 2500 *request* anomali dan 2500 *request* normal URL. [7] Berikut table dataset tersebut.

Tipe	*Training Set*	*Test Set*	Total
Normal	2000	500	2500
Anomali	2000	500	2500

Tabel 2. WAF dataset

4.2 PERFORMANCE MEASUREMENT

Performa model dapat diukur menggunakan *accuracy*, *sensitivity*, *specificity*. *Accuracy* merupakan kemampuan model dalam mengklasifikasikan data dengan benar. Tetapi, *accuracy* tidak dapat mengetahui kesalahan *request* anomali tetapi tidak terdeteksi. Untuk itu, digunakan pengukuran *sensitivity* merupakan kemampuan model dalam mengklasifikasi *request* anomali dengan benar dari semua *request* anomali. Dan, *specificity* merupakan kemampuan model dalam mengklasifikasi *request* normal dengan benar dari semua *request* normal.

$$Accuracy = \frac{Total\ data\ diklasifikasi\ benar}{Total\ data}$$

$$Sensitivity = \frac{Total\ data\ anomali\ diklasifikasi\ benar}{Total\ data\ anomali}$$

$$Specificity = \frac{Total\ data\ normal\ diklasifikasi\ benar}{Total\ normal}$$

4.3 RESULT

Dengan melakukan tes terhadap model yang telah dibuat dapat dilihat hasilnya. [7] Pada tabel 3, merupakan hasil dari tes model menggunakan dataset CSIC. Hasil *sensitivity* terbaik dari model yang diberikan adalah 97% dengan menggunakan LSTM. Sehingga, menandakan bahwa kemampuan dalam mendeteksi anomali sangat baik. Selain itu, hasil

dari *specificity* memberikan hasil kesalahan deteksi *request* normal yang hampir mendekati nol. Jadi, penggunaan model menggunakan LSTM memberikan performa yang terbaik pada dataset CSIC.

Metode	*Accuracy*	*Sensitivity*	*Specificity*
Model tanpa RNN	0.8515	0.7403	0.9546
Model dengan GRU	0.9788	0.9722	0.9845
Model dengan LSTM	**0.9842**	**0.9756**	**0.9921**

Tabel 3. Hasil Performa pada CSIC dataset

Pada tabel 4, merupakan hasil dari tes model menggunakan dataset WAF. Hasil *sensitivity* terbaik dari model dengan menggunakan metode GRU adalalah 98%. Tidak hanya itu, metode GRU juga memberikan hasil yang terbaik dengan pengukuran secara *accuracy* dan *specificity*. Jadi, penggunaan model menggunakan GRU memberikan performa terbaik pada dataset WAF.

Metode	*Accuracy*	*Sensitivity*	*Specificity*
Model tanpa RNN	0.9475	0.9462	0.9488
Model dengan GRU	**0.9856**	**0.9880**	**0.9832**
Model dengan LSTM	0.9838	0.9858	0.9818

Tabel 3. Hasil Performa pada CSIC dataset

Pada kolom metode tabel diatas, ada metode dengan menggunakan model tanpa RNN dan menggunakan RNN-GRU serta RNN-LSTM. Performa yang diberikan oleh model tanpa RNN lebih rendah jika dibandingkan dengan model RNN. Sehingga, penggunaan RNN pada model ini sangat penting dan berpengaruh sekali. Performa dari RNN-LSTM adalah yang terbaik di dataset CSIC dan RNN-GRU adalah yang terbaik di dataset WAF. Performa metode kedua RNN tersebut dapat dikatakan seimbang. Jadi, penggunaan RNN pada model memberikan performa yang lebih baik.

5. KESIMPULAN

Dari penjelasan diatas, dapat diambil kesimpulan penggunaan RNN untuk deteksi serangan terhadap aplikasi web menggunakan metode *anomaly-based*. RNN yang digunakan pada model yaitu, GRU dan LSTM. RNN dilatih untuk mempelajari pola *request* normal. Dan dengan menggunakan MLP untuk memprediksi *request* normal atau anomali berdasarkan hasil *training*

dari RNN. Dengan metode tersebut dilakukan *testing* menggunakan dataset *request* normal dan anomali yang tersedia yaitu, CSIC dan WAF dataset. Hasil pengujian menggunakan metode tersebut mendapatkan performa yang sangat baik dengan pengukuran *accuracy*, *sensitivity*, dan *specificity*. Jadi, penggunaan RNN mampu dalam deteksi serangan terhadap aplikasi web.

6. REFERENSI

[1] Z. Tian, C. Luo, J. Qiu, X. Du and M. Guizani, "A Distributed Deep Learning System for Web Attack Detection on Edge Devices," in *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963-1971, March 2020.

[2] Positive Technology, "Web Applications vulnerabilities and threats: statistics for 2019," February 13, 2020. [Online]. Available at https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/

[3] Y. Pan, F, Sun, Z. Teng. "Detecting web attacks with end-to-end deep learning," *Journal Internet Service Application* 10, 16 (2019). [Online]. Available at https://jisajournal.springeropen.com/articles/10.1186/s13174-019-0115-x

[4] L. Duc, A. Zincir-Heywood, H. Malcolm, "Unsupervised Monitoring of Network and Service Behaviour Using Self Organizing Maps," *Journal of Cyber Security and Mobility*, 2018. [Online]. Available at 10.13052/jcsm2245-1439.812

[5] T. Katte, "Recurrent Neural Network and its Various Architecture Types," in *International Journal of Research and Scientific Innovation*, Volume V. IJRSI. March 2018.

[6] S. Hojjat, S. Sharan, B. Joseph, C. Errol, V. Shahrokh, "Recent Advances in Recurrent Neural Network," 2017. *arXiv :1801.01078v3* (2018).

[7] J. Liang, W. Zhao, W. Ye, "Anomaly-Based Web Attack Detection: A Deep Learning Approach," in *International Conference on Network, Communication, and Computing*, pp. 80-85. ICNCC, 2017.

[8] K. Daljit, K. Parminder, "Empirical Analysis of Web Attacks," in *International Conference on Information Security & Privacy*, December 2015.

[9] C. Gimenez, A. Villegas, G. Maranon, "HTTP Dataset CSIC 2010," in *Spanish Research National Council*, 2012. [Online]. Available at http://www.isi.csic.es/dataset/

[10] I. Corona, D. Ariu, and G. Giacinto, "A framework for the detection of attacks against web applications," in *IEEE International Conference*, pp. 1–6. IEEE, 2009.

[11] K. Christopher, V. Giovanni, "Anomaly Detection of Web-based Attacks," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2003.