BAB IV. HASIL PENELITIAN DAN PEMBAHASAN

4.1. Hasil Penelitian

Pemaparan hasil penelitian dibagi menjadi dua yaitu terkait dengan (1) metode deteksi *cyber attack* menggunakan *machine learning* dan (2) metode mitigasi *cyber attack* menggunakan *multi-layer security.* Adapun pemaparan hasil penelitian terse-but disampaikan pada sub-bab 4.1.1 dan 4.1.2 berikut ini.

4.1.1. Metode Deteksi Menggunakan *Machine Learning*

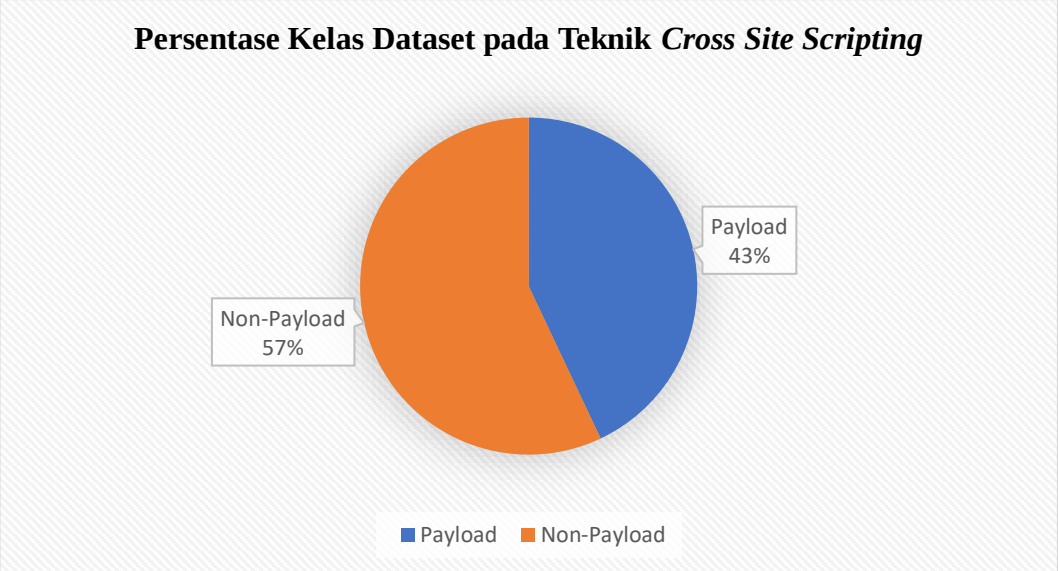
Seperti yang telah diungkapkan pada Bab II dan III bahwa untuk meng-optimalkan dan meningkatkan tingkat akurasi metode deteksi, eksperimen mengu-jikan lima al-goritma *machine learning*, yaitu (1) *Naïve Bayes*; (2) *Logistic Regre-ssion*; (3) *Gradient Boosting; (4) Support Vector Machine;* dan (5) *K-Nearest Nei-ghbor.* Kelima algoritma tersebut diujikan pada masing-masing dataset tiga teknik serangan yang berbeda-beda yaitu XSS, SQLi, dan RCE. Berdasarkan hasil pengu-jian yang telah dilakukan, algoritma yang memiliki tingkat akurasi tinggi dan waktu proses rendah dalam mendeteksi serangan siber adalah ***Support Vector Machine.*** Berikut adalah hasil implementasi metode deteksi pada tiap teknik serangan.

4.1.1.1. Teknik Serangan *Cross Site Scripting* (XSS)

1) Karakteristik Dataset

Metode deteksi pada teknik serangan XSS menggunakan dua dataset dengan total baris dataset sebanyak 49.823. Jumlah baris dataset pertama adalah 43.217 dan

dataset kedua adalah 6.606. Setelah dilakukan *data preprocessing*, total baris data-set yang *eligble* untuk dijadikan *data training* dan *testing* adalah 49.226. Dari total baris dataset *eligible* tersebut, total dataset dengan label *payload* adalah 21.158 dan *non-payload* adalah 28.068 (tertuang pada gambar 4.1).



Gambar 4.1 Persentase Kelas Dataset pada Teknik Serangan XSS

2) Konfigurasi Parameter Deteksi

Setelah dilakukan 15 kali tahapan optimalisasi dan evaluasi kinerja, terutama pada konfigurasi parameter *margin* dan *corpus rules*, tingkat akurasi tinggi dan ToP (*Time of Process)* yang rendah didapatkan dengan parameter konfigurasi berikut.

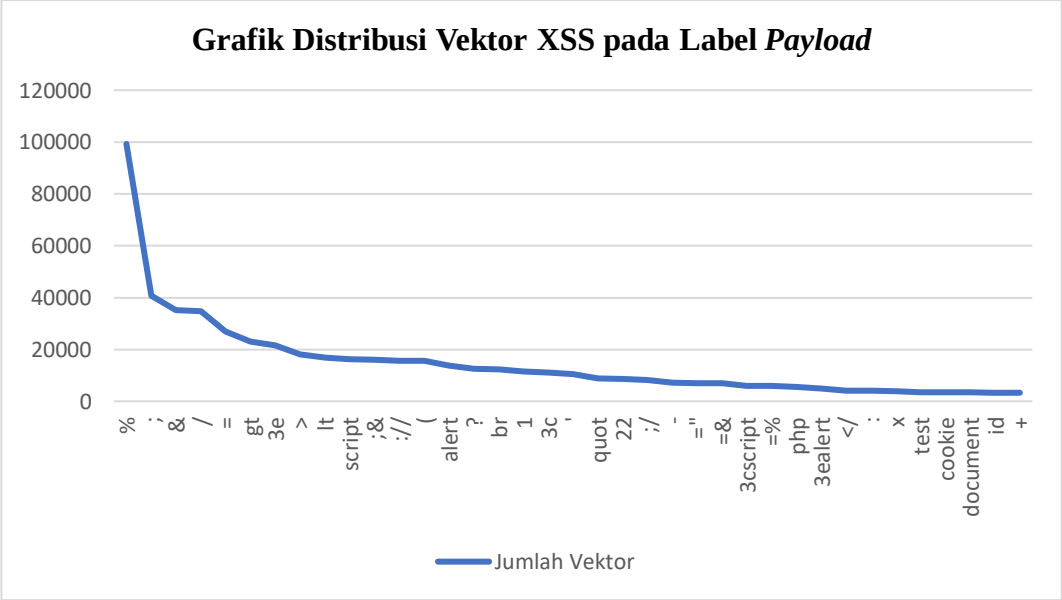
Tabel 4.1. Konfigurasi Parameter Optimal pada Serangan XSS

No	Parameter	Nilai
1	*Margin*	5
2	*Test Size*	0.2
3	*Lowercase Conversion*	*True*
4	*Alphanumeric Filter*	*False*
5	*Punctuation Remover*	*False*

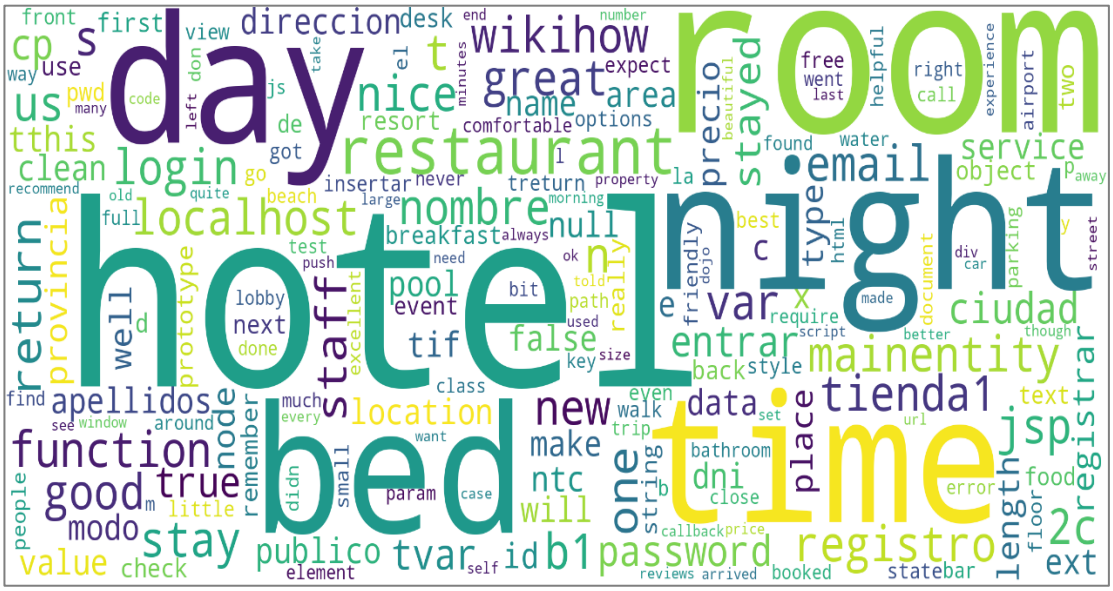
3) Feature Set Injection

Dengan batas vektor sebesar 505, penciri setiap baris dataset diambil dari masing-masing label, baik *payload* maupun *non-payload*, kemudian dihimpun ke dalam variabel *feature set*. Tahapan ini disebut sebagai *feature set injection*, yaitu tahapan ketika penciri label *payload* dan *non-payload* diekstrak dan dihimpun menjadi satu set atau seperangkat fitur (gambar 4.2 s.d 4.5).

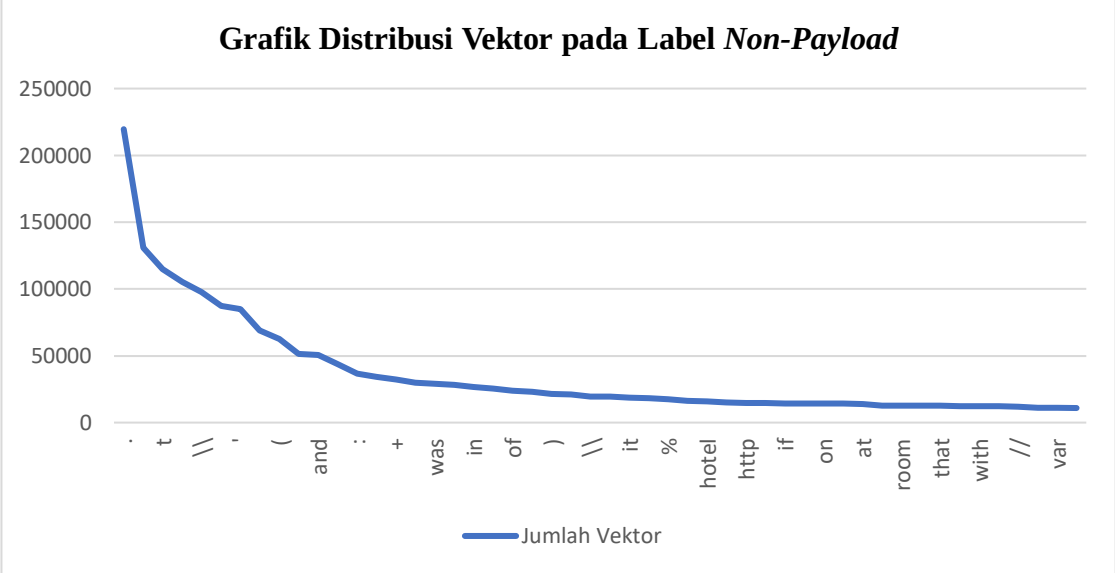
Gambar 4.2. Visualisasi Vektor XSS Label *Payload* Menggunakan *Wordcloud*



Grafik 4.1. Grafik Distribusi Vektor XSS pada Label Payload



Gambar 4.3. Visualisasi Vektor XSS Label *Non-Payload* Menggunakan *Wordcloud*



Grafik 4.2. Grafik Distribusi Vektor XSS pada Label *Non-Payload*

4) Tingkat Akurasi dan Efesiensi ToP

Setelah dilakukan 15 eksperimen pada masing-masing algoritma dengan konfigurasi parameter yang berbeda-beda, algoritma SVM terbukti memiliki tingkat akurasi dan ToP yang baik dibandingkan dengan algoritma yang lain. SVM mampu mendapatkan tingkat akurasi 0,996546 dengan ToP 3 menit 46 detik.

Dengan ToP tersebut, kecepatan algoritma ini mencapai 0,009618027 *micro seconds* untuk tiap *query*. Tingkat akurasi dan ToP algoritma adalah sebagai berikut.

Tabel 4.2 Tingkat Akurasi dan ToP *Classifier* dalam Serangan XSS

No	Nama *Classifier*	Tingkat Akurasi	*ToP of Learning*	ToP
1	Support Vector Machine	0,996546821	0:03:46.050874	0,009618027
2	K-Nearest Neigbors	0,995835872	0:21:38.650214	0,544929926
3	Logistic Regression	0,994718667	0:00:40.664987	0,000824402
4	Gradient Boosting	0,992281129	0:01:53.887530	0,001007603
5	Naive Bayes	0,703229738	0:00:44.060281	0,002381607

5) *Confussion Matrix*, Visualisasi Akurasi, dan ToP

Seperti yang telah tertera pada tabel 4.2, SVM menjadi pilihan algoritma metode deteksi karena mampu mencapai tingkat akurasi tertinggi, yaitu 0,9965468 dengan ToP sebesar 0,009618027 mikro detik per *query*, dibandingkan dengan alternatif algoritma yang lain. Dengan tingkat akurasi dan ToP tersebut, SVM dapat melakukan deteksi serangan XSS lebih akurat dan lebih cepat dibandingkan dengan KNN, *Logistic Regression, Gradient Boosting*, dan *Naïve Bayes*. Terkait dengan tingkat akurasi SVM tersebut, berikut ini adalah data-data *confussion matrix* pada metode deteksi serangan XSS menggunakan SVM.

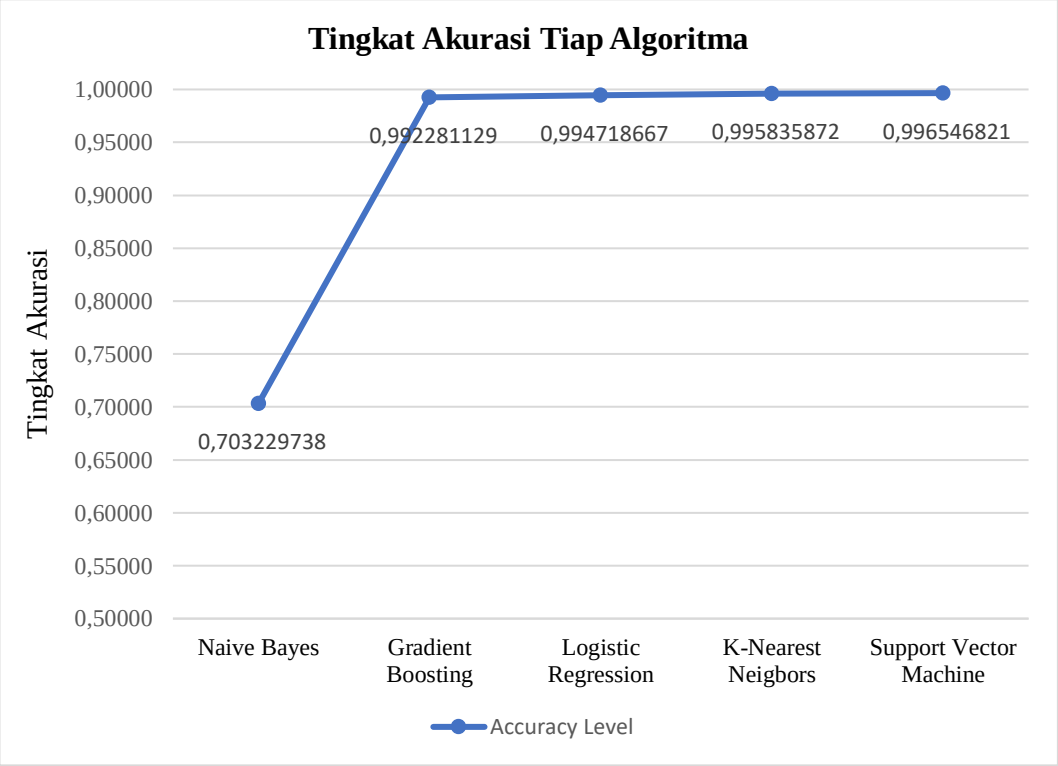
Tabel 4.3. *Confusion Matrix* Deteksi Serangan XSS Menggunakan SVM

No	Label	*Non-Payload*	*Payload*
1	*Non-Payload*	<5573>	15
2	*Payload*	19	<4239>

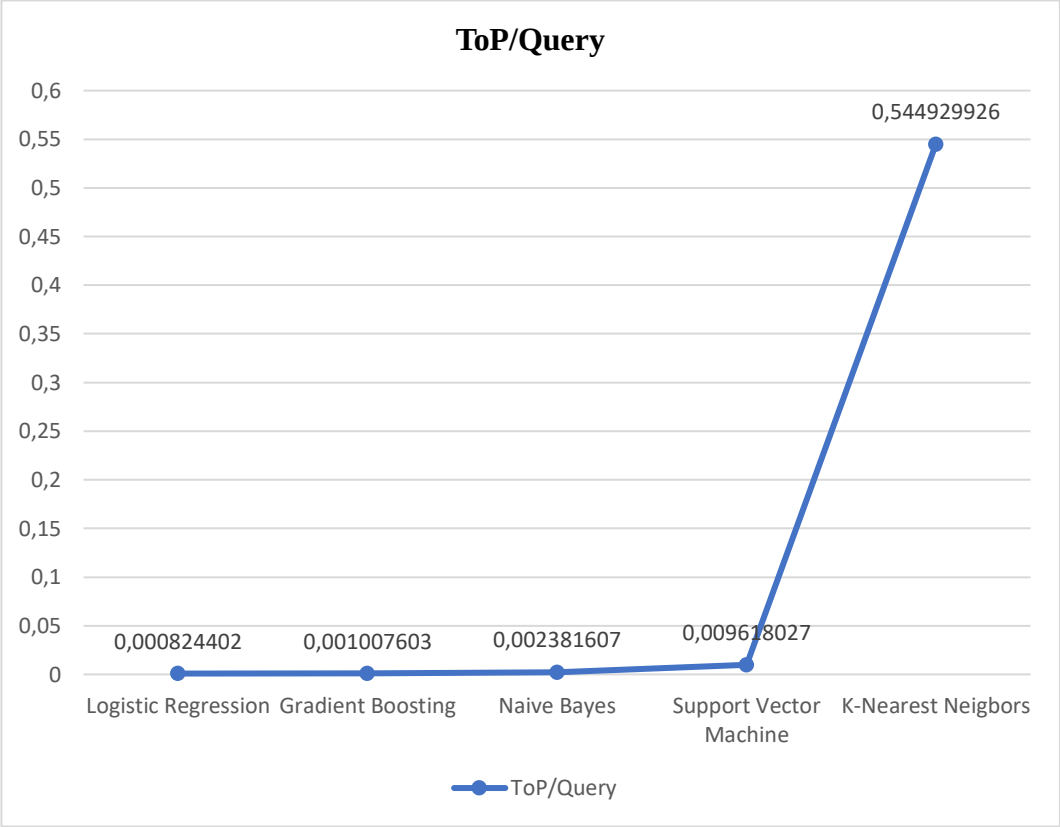
Tabel 4.4. Rincian *Precision, Recall,* dan *F-Measure* Deteksi XSS

No	Label	*Precision*	*Recall*	*F-Measure*
1	*Non-Payload*	0.996602	0.997316	0.996959
2	*Payload*	0.996474	0.995538	0.996006

Jika dilihat dari nilainya, tingkat akurasi setiap algoritma cukup bervariasi. Naïve Bayes menjadi algoritma yang mendapatkan tingkat akurasi terkecil, yaitu hanya 0,703229. Dengan tingkat akurasi tersebut, posisi tingkat akurasi dari Naïve Bayes cukup jauh dibandingkan algoritma lainnya. Terlihat pada grafik 4.3, jarak algoritma lain seperti SVM, KNN, *Gradient Boosting*, dan *Logistic Regression* tidak terlalu jauh atau signifikan. Sementara itu, pada grafik 4.4, KNN menjadi algoritma yang memerlukan ToP lebih lama dibandingkan lainnya. Jarak ToP KNN cukup signifikan dibandingkan algoritma yang lain.



Grafik 4.3. Tingkat Akurasi Tiap Algoritma dalam Mendeteksi Serangan XSS

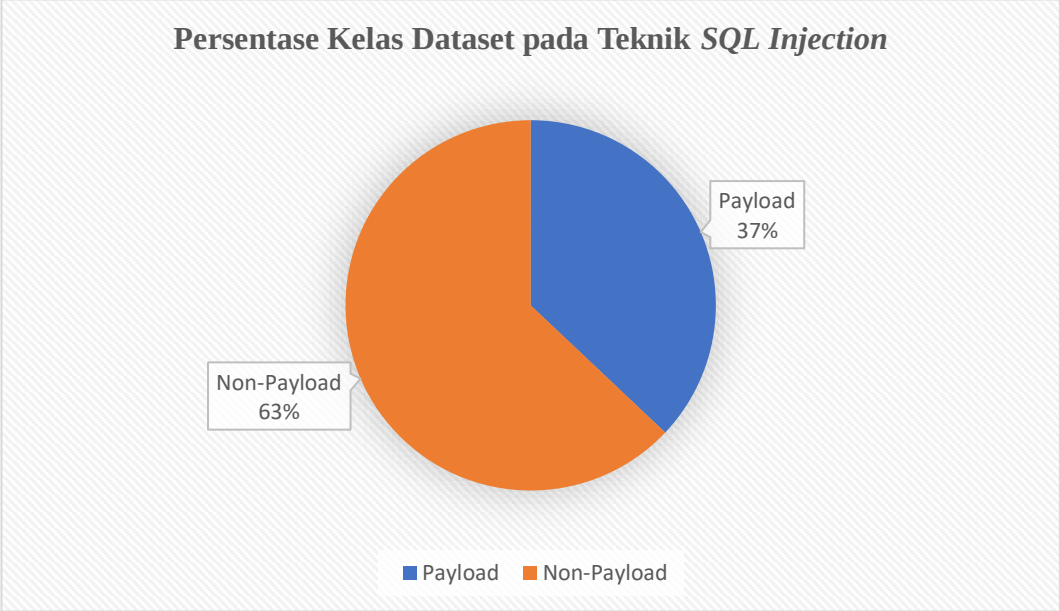


Grafik 4.4. ToP Tiap Algoritma dalam Mendeteksi Serangan XSS

4.1.1.2. Teknik Serangan *SQL Injection* (SQLi)

1) Karakteristik Dataset

Jumlah keseluruhan dataset SQLi yang digunakan adalah 30.904 baris dataset. Setelah dilakukan *data preprocessing*, terdapat 295 baris data yang dieliminasi sehingga jumlah data *eligible* yang digunakan dalam proses *data training* dan *data testing* adalah sebanyak 30.609. Sama halnya dengan dataset pada teknik serangan XSS, terdapat dua label yang digunakan pada dataset SQLi yaitu *payload* dan *non-payload*. Dari total 30.906 baris dataset tersebut, jumlah baris data dengan label payload adalah 11.341 dan data dengan label *non-payload* sebesar 19.268. Persentase kedua label pada dataset SQLi adalah sebagai berikut.



Gambar 4.4. Persentase Kelas Dataset pada Teknik Serangan *SQL Injection*

2) Konfigurasi Parameter Deteksi

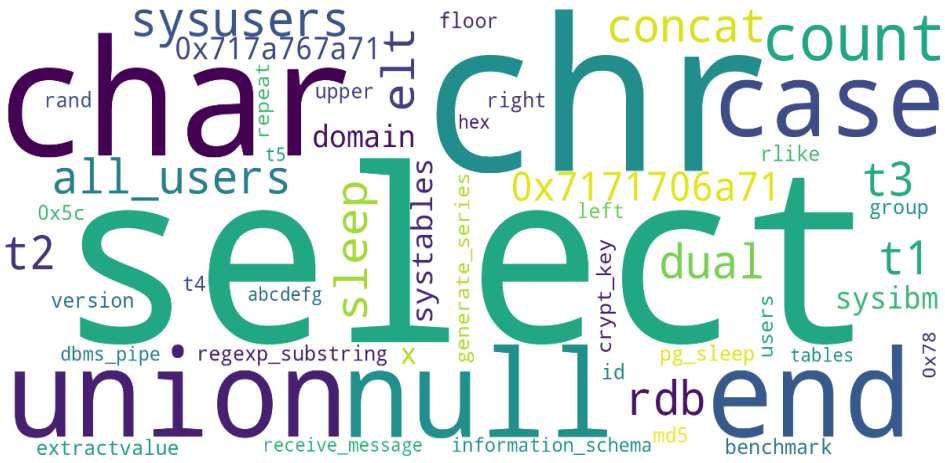
Optimalisasi dan evaluasi kinerja metode deteksi pada teknik serangan SQLi dilakukan dalam 20 kali tahapan. Tiap tahapan mengujikan lima algoritma. Dengan kata lain, terdapat 100 (20 tahapan x 5 algoritma) kali pengujian algoritma yang telah dilakukan dengan konfigurasi parameter yang berbeda-beda. Proses ini dilakukan melalui pengaturan konfigurasi parameter *margin* dan *corpus rules*. Adapun parameter konfigurasi deteksi SQLi dirincikan pada tabel 4.7 berikut ini.

Tabel 4.5 Konfigurasi Parameter Optimal pada Serangan SQLi

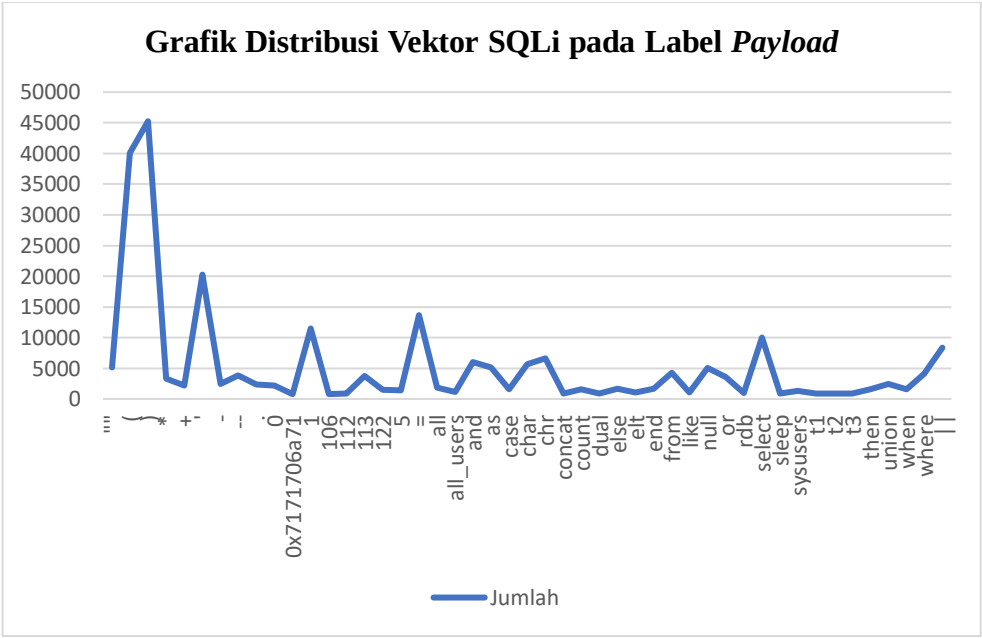
No	Parameter	Nilai
1	*Margin*	6.5
2	*Test Size*	0.2
3	*Lowercase Conversion*	*True*
4	*Alphanumeric Filter*	*False*
5	*Punctuation Remover*	*False*

3) Feature Set Injection

Pada serangan SQLi, jumlah batas maksimal vektor dengan *margin* 6.5 adalah sebanyak 104 vektor. Pada tahapan *feature set injection*, sebanyak 104 vektor pen-ciri pada label *payload* dan *non-payload* dihimpun menjadi seperangkat fitur. *Feature set injection* yang digunakan untuk mendeteksi serangan SQLi dapat dilihat pada gambar 4.5 sampai dengan 4.6.



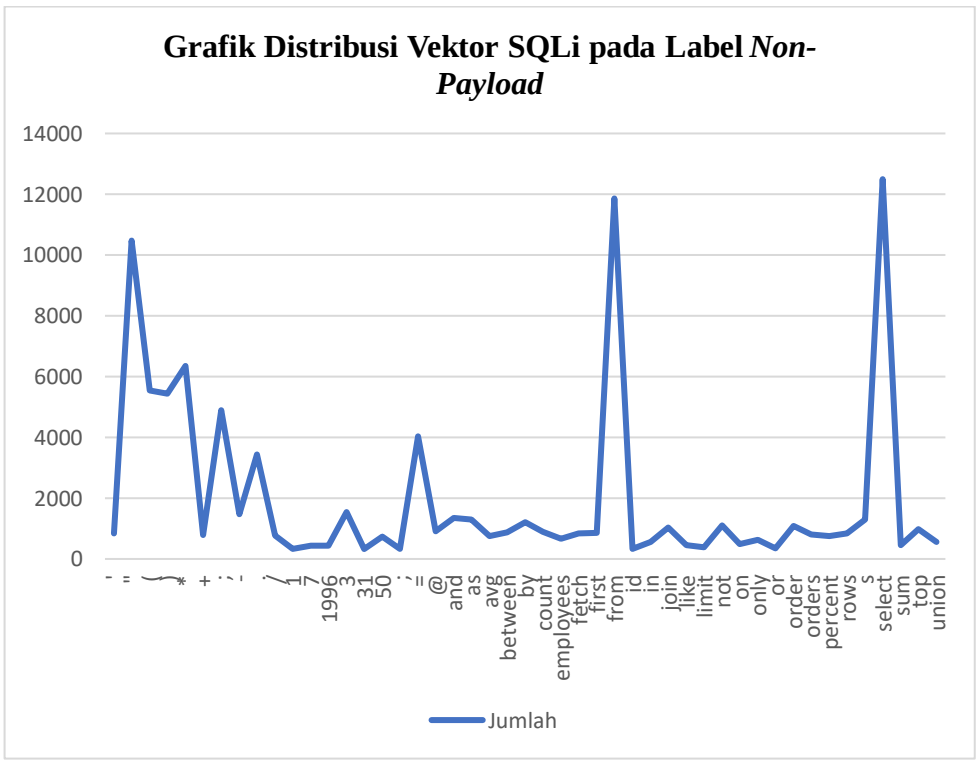
Gambar 4.5. Visualisasi Vektor SQLi Label *Payload* Menggunakan *Wordcloud*



Grafik 4.5. Grafik Distribusi Vektor SQLi pada Label *Payload*

Gambar 4.6 menampilkan daftar vektor yang digunakan dalam melakukan serangan SQLi. Masih terkait dengan gambar 4.6, gambar 4.7 menampilkan 50 vektor yang disertai dengan jumlah kemunculannya di dalam baris dataset. Sementara itu, vektor yang digunakan pada label *payload* dapat dilihat pada gambar 4.8 dan 4.9

Gambar 4.6. Visualisasi Vektor SQLi Label *Non-Paylod* Menggunakan *Wordcloud*



Grafik 4.6. Grafik Distribusi Vektor SQLi pada Label *Non-Payload*

4) Tingkat Akurasi dan Efesiensi ToP

Berdasarkan eksperimen yang telah dilakukan, SVM menjadi algoritma yang mampu mendeteksi dengan tingkat keakuratan tertinggi, yaitu 0,99771. Selain itu, ToP yang dihasilkan juga sangat cepat karena masih berada di bawah mikro detik, yaitu 0,00100 mikro detik per *query*. Dengan tingkat akurasi dan ToP ini, SVM terbukti dapat diandalkan dalam mendeteksi serangan SQLi.

Tabel 4.6. Tingkat Akurasi dan ToP Classifier dalam Serangan SQLi

	Classifier Name	Accuracy	ToP of Learning	ToP/Query
1	Support Vector Machine	0,997713166	0:00:10.713473	0,001008
2	K-Nearest Neigbors	0,997059784	0:01:23.933334	0,055445
3	Logistic Regression	0,996079713	0:00:04.485225	0,000444
4	Gradient Boosting	0,99477295	0:00:13.379335	0,000652
5	Naive Bayes	0,975498203	0:00:04.138092	0,000415

5) *Confussion Matrix*, Visualisasi Akurasi, dan ToP

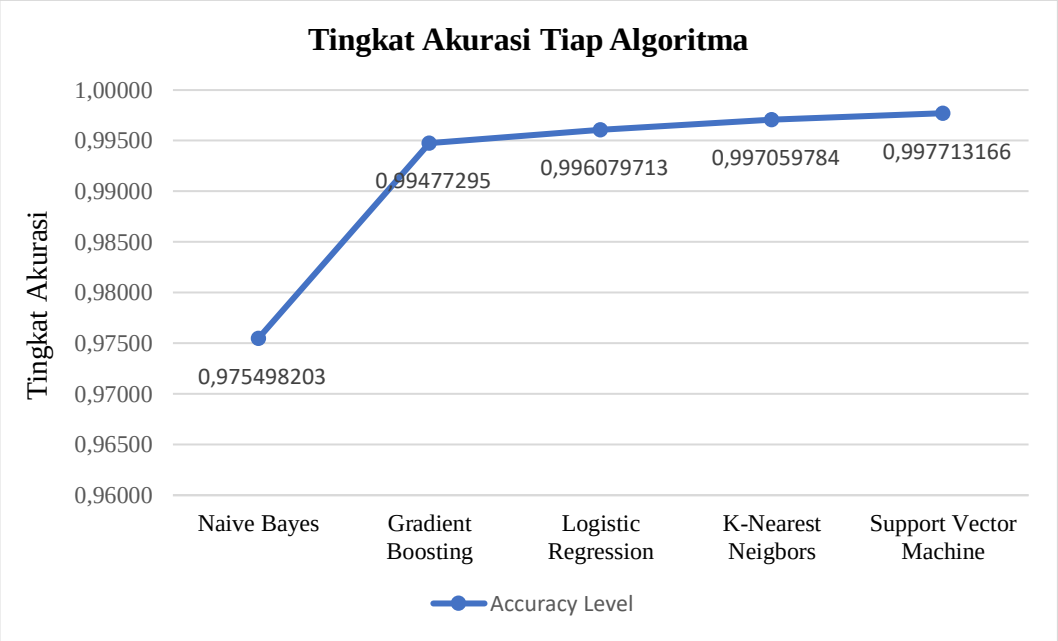
SVM mampu mencapai tingkat akurasi tertinggi dibandingkan algoritma lain-nya. Dengan tingkat akurasi 0,997713, SVM mampu melakukan deteksi serangan SQLi lebih akurat. Selain itu, ToP algoritma ini juga terbilang cepat karena hanya memerlukan 0,00100 mikro detik. Berikut ini adalah data *confussion matrix* pada metode deteksi serangan SQLi menggunakan SVM.

Tabel 4.7. *Confusion Matrix* Deteksi Serangan SQLi Menggunakan SVM

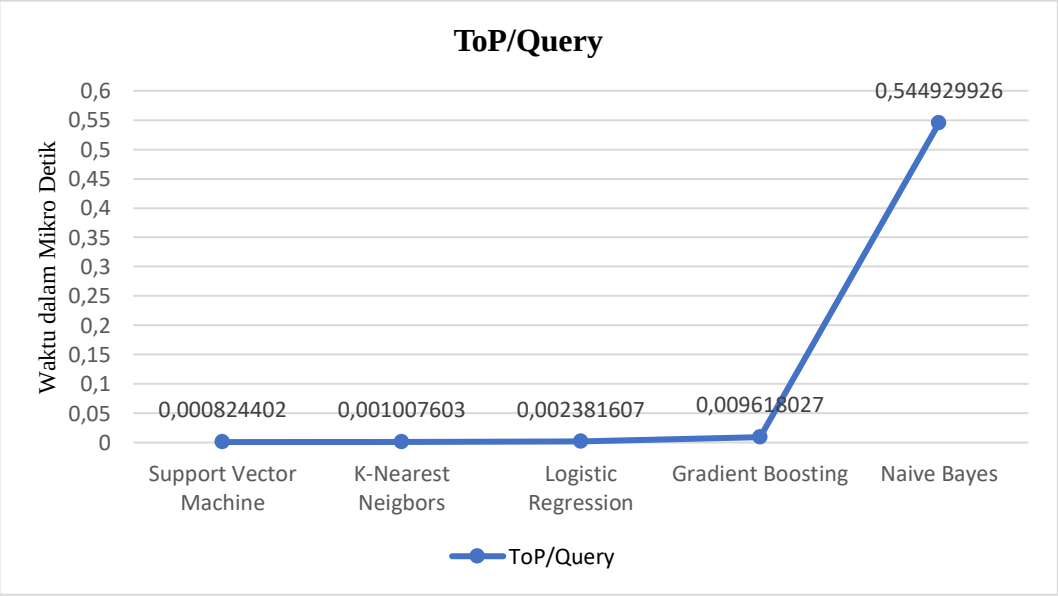
No	Nama *Classifier*	*Non-Payload*	*Payload*
1	*Non-Payload*	<3890>	1
2	*Payload*	13	<2218>

Tabel 4.8. Rincian *Precision*, *Recall*, dan *F-Measure* Deteksi SQLi

No	Label	*Precision*	*Recall*	*F-Measure*
1	*Non-Payload*	0.996669	0.999743	0.998204
2	*Payload*	0.999549	0.994173	0.996854



Gambar 4.7. Tingkat Akurasi Tiap Algoritma dalam Mendeteksi Serangan SQLi



Grafik 4.7 ToP Tiap Algoritma dalam Mendeteksi Serangan XSS

4.1.1.3. Teknik Serangan *Remote Code Execution* (RCE)

1) Karakteristik Dataset

Sedikit berbeda dengan dataset serangan XSS dan SQLi yang berbasis CSV atau dalam format tabel, dataset RCE pada penelitian ini terdiri dari dua jenis dataset, yaitu (1) berbasis direktori dan (2) berbasis tabel. Dataset berbasis tabel adalah dataset dengan label *payload*, sedangkan yang berbasis tabel adalah dataset dengan label *non-payload*. Oleh karena itu, pada tahapan *data preprocessing* juga dilaku-kan kompilasi dataset *payload* dan *non-payload*.

Untuk menghimpun dataset RCE label *payload* pada masing-masing direktori, peneliti melakukan *directory crawling*. Pada tahapan ini, dataset RCE format teks dibaca dan dihimpun ke dalam satu variabel dataset *payload*. Jumlah berkas dataset yang diindeks sebanyak 19 berkas, yang berada pada tiga direktori. Setiap berkas dikonversi atau di-*parsing* sesuai dengan karakteristiknya. Setelah dikonversi menjadi tabel atau *data frame*, jumlah baris dataset label *payload* yang digunakan pada penelitian adalah 24.095 baris dataset. Setelah semua dataset *payload* dikompilasi, dataset yang diproses adalah baris dataset *non-payload*.

Sementara itu, dataset dengan label *non-payload* diambil dari dataset publik dengan format tabel atau CSV. Jumlah baris dataset *non-payload* adalah 24.097. Dengan demikian, jumlah dataset label *payload* dan *non-payload* yang digunakan pada penelitian sebesar 48.192 baris dataset. Setelah dilakukan *data preprocessing* pada kedua dataset tersebut, jumlah baris dataset yang *eligble* untuk digunakan pada proses *data training* dan *data testing* adalah 47.177, yang artinya terdapat 1015 baris dataset yang tidak dipergunakan atau dielaminasi.

2) Konfigurasi Parameter Deteksi

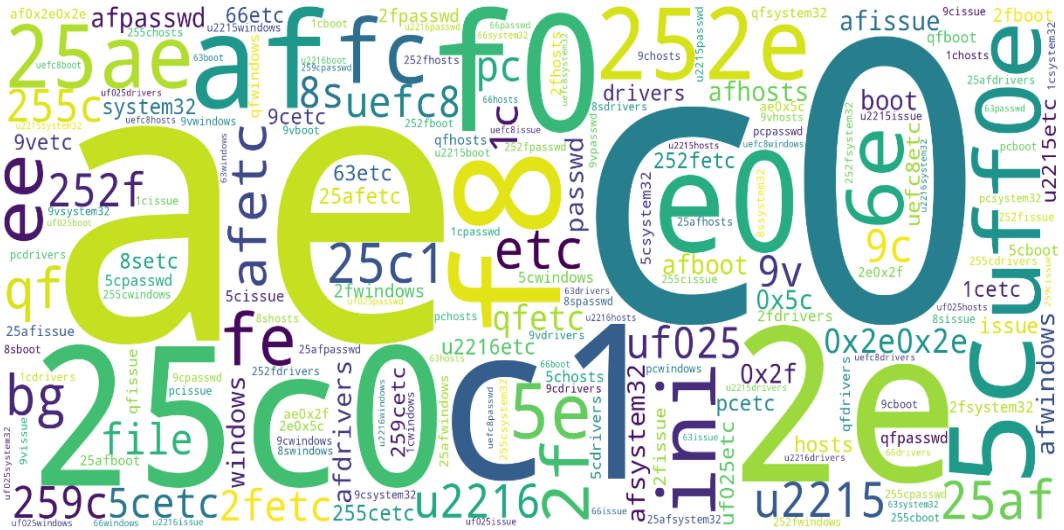
Konfigurasi parameter deteksi diujikan dalam 18 kali tahapan eksperimen. Tiap tahapan eksperimen mengujikan lima algoritma, sehingga total eksperimen yang dilakukan sebanyak 90 eksperimen (18 tahapan x 5 algoritma). Selain itu, proses *data training* dilakukan sebanyak 10 s.d 30 kali untuk mendapatkan performa dan tingkat akurasi yang tertinggi. Tidak hanya melakukan *comprehensive data learning, parameter configuration switching* metode deteksi juga dilakukan. Tahapan ini dilakukan untuk mendapatkan konfirgurasi parameter yang paling tepat. Adapun konfigurasi parameter untuk mendeteksi RCE adalah sebagai berikut.

Tabel 4.9 Konfigurasi Parameter Optimal pada Serangan RCE

No	Parameter	Nilai
1	*Margin*	15
2	*Test Size*	0.2
3	*Lowercase Conversion*	*True*
4	*Alphanumeric Filter*	*False*
5	*Punctuation Remover*	*False*

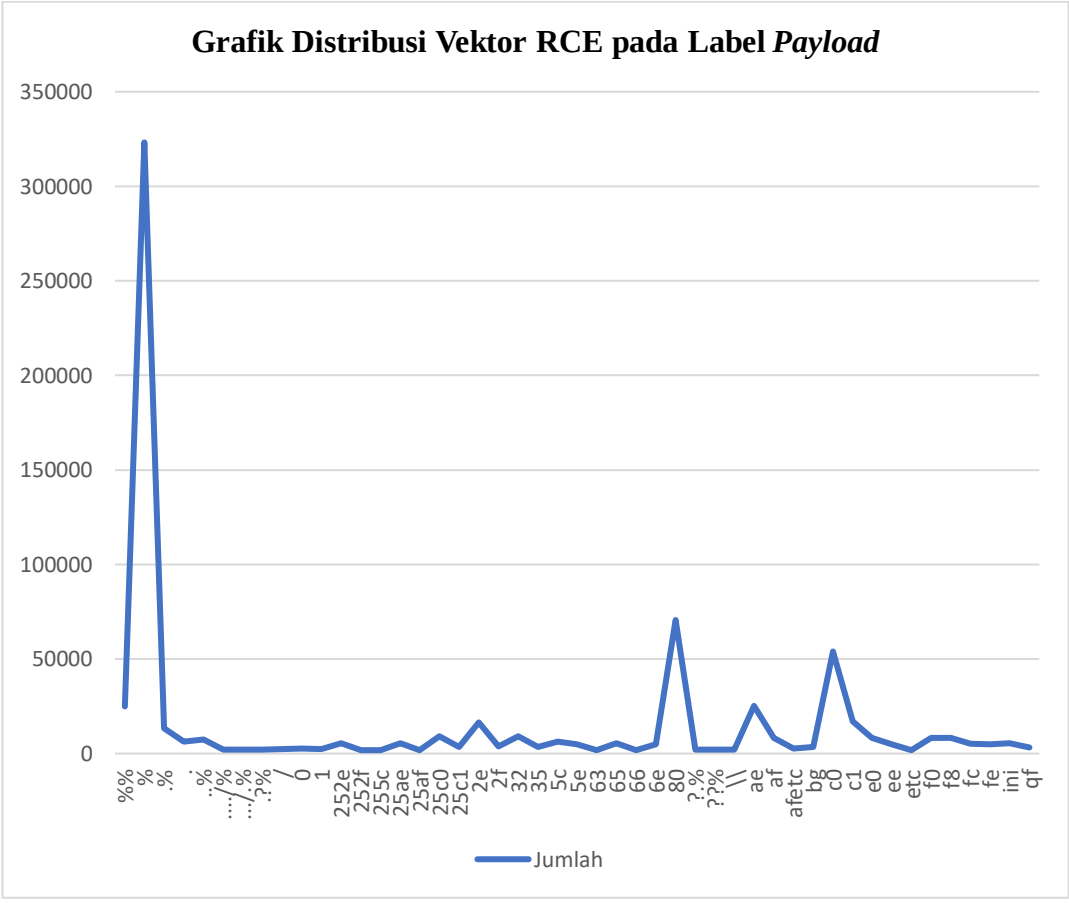
3) *Feature Set Injection*

Setelah dilakukan 90 eksperimen, konfirgurasi margin yang paling optimal berada pada angka 15. Setelah dikalkulasikan dengan margin, batas vektor metode deteksi serangan RCE menjadi 365. Seperti pada serangan XSS dan SQLi, 365 vektor pada label *payload* dan *non-payload* digunakan sebagai parameter untuk mendeteksi serangan RCE. Penentuan batas vektor merupakan hal penting, karena berpengaruh pada tingkat akurasi metode deteksi dan peforma model yang dihasilkan. Berikut ini adalah vektor yang digunakan pada serangan RCE.



Gambar 4.8. Visualisasi Vektor RCE Label *Payload* Menggunakan *Wordcloud*

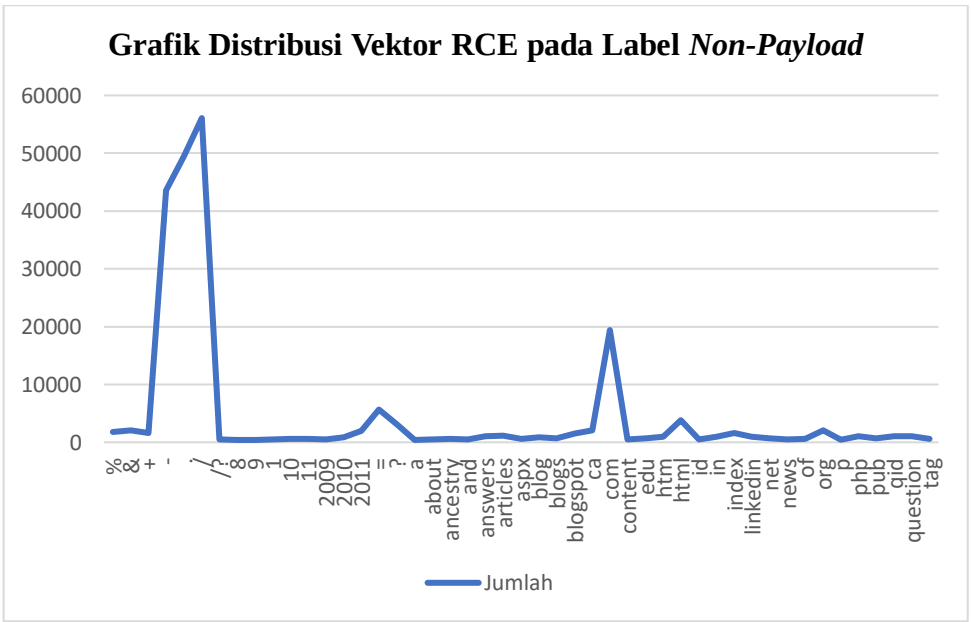
Gambar di atas menunjukan vektor-vektor yang digunakan untuk mendeteksi serangan RCE. Sementara itu, jumlah distribusi tiap vektor adalah sebagai berikut.



Grafik 4.8. Grafik Distribusi Vektor RCE pada Label *Payload*

Serangan RCE melalui aplikasi website mayoritas dikirimkan melalui metode GET dan POST. *Website* dengan *modern architechture* sekalipun tidak terlepas dari penerapan metode GET dan POST. Oleh karena itu, baik melalui metode GET ataupun POST, metode deteksi harus mampu mengenali URL atau *query non-payload*. Terkait dengan hal tersebut, berikut ini adalah daftar vektor dan distribusi vektor pada label *non-payload* yang digunakan pada teknik serangan RCE.

Gambar 4.9. Visualisasi Vektor RCE Label *Payload* Menggunakan *Wordcloud*



Grafik 4.9 Grafik Distribusi Vektor RCE pada Label *Non-Payload*

4) Tingkat Akurasi dan Efisiensi ToP

Berdasarkan proses eksperimen dan optimalisasi yang telah dilakukan, SVM mam-pu mendapatkan tingkat akurasi tertinggi, yaitu 0,987495 dengan ToP sebe-sar 0,004356529 mikro detik. Dengan ToP tersebut, model yang dihasilkan dapat melakukan deteksi serangan RCE dengan *delay* relatif rendah. Berikut ini adalah rincian masing-masing tingkat akurasi dan ToP pada masing-masing algoritma.

Tabel 4.10. Tingkat Akurasi dan ToP *Classifier* dalam Serangan XSS

	Classifier Name	Accuracy	ToP	ToP/Query
1	Support Vector Machine	0,987495	0:01:21.085515	0,004356529
2	K-Nearest Neigbors	0,985375	0:08:19.264679	0,380548687
3	Logistic Regression	0,984209	0:00:24.859991	0,000706464
4	Gradient Boosting	0,980606	0:01:01.514690	0,000824208
5	Naive Bayes	0,97446	0:00:24.987471	0,001295184

5) *Confussion Matrix*, Visualisasi Akurasi, dan ToP

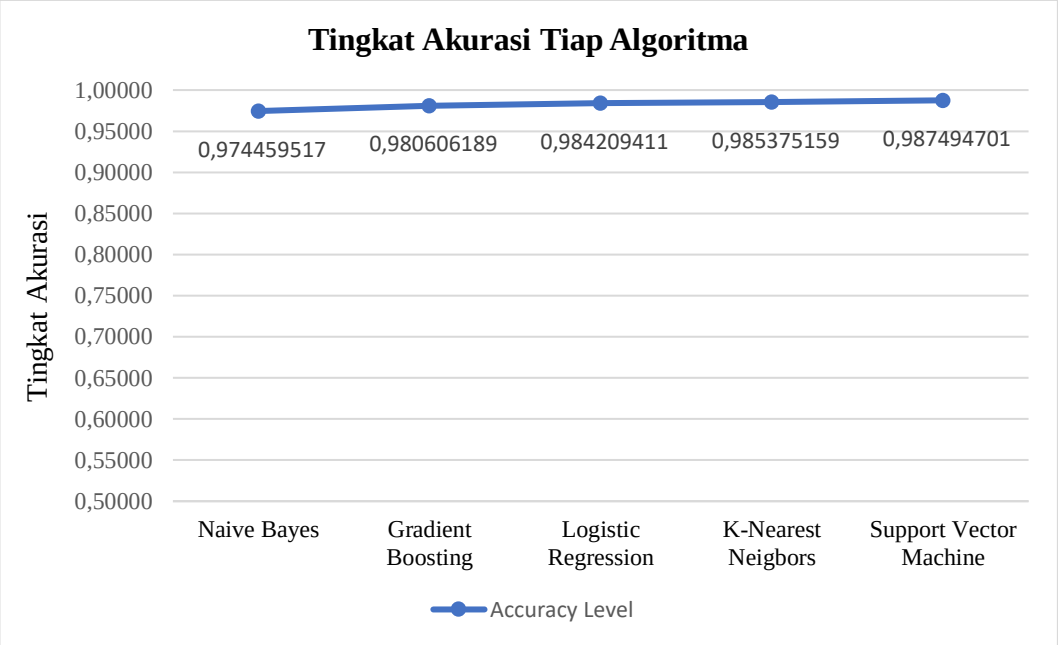
Confussion matrix metode deteksi serangan RCE menggunakan SVM dapat dilihat pada tabel 4.12. Sesuai dengan data yang tertera pada tabel tersebut, SVM mampu mendeteksi secara benar 4820 *non-payload* dan 4498 *payload*. Berikut ini adalah rincian *confussion matrix* metode deteksi serangan RCE menggunakan SVM, ter-masuk rincian *precision, recall,* dan *f-measure*.

Tabel 4.11. *Confusion Matrix* Metode Deteksi Serangan RCE Menggunakan SVM

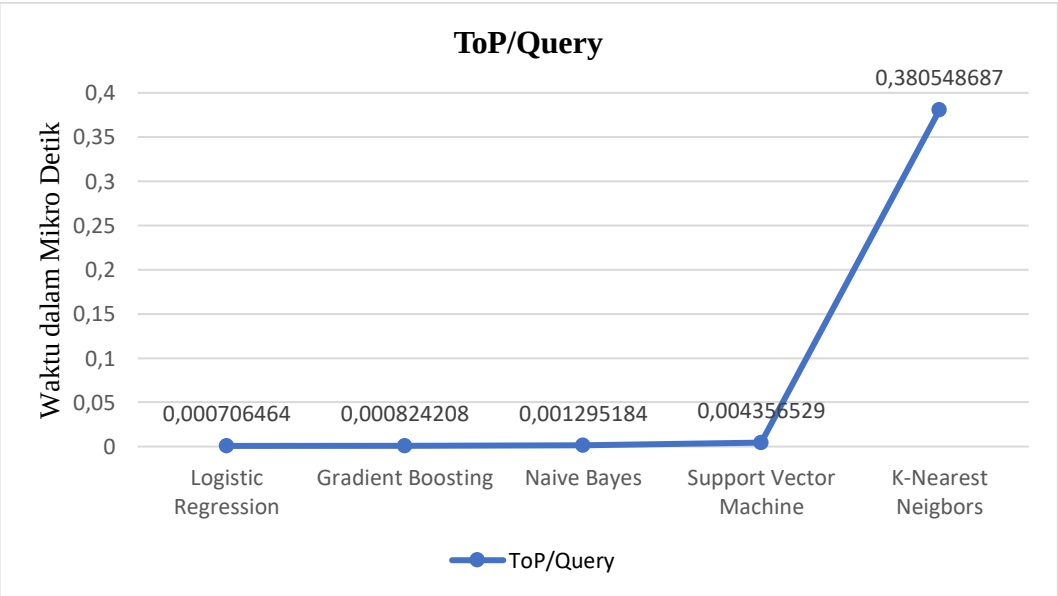
No	Nama *Classifier*	*Non-Payload*	*Payload*
1	*Non-Payload*	<4820>	35
2	*Payload*	83	<4498>

Tabel 4.12. Rincian *Precision, Recall,* dan *F-Measure* Deteksi RCE

No	Label	*Precision*	*Recall*	*F-Measure*
1	*Non-Payload*	0.983072	0.983072	0.983072
2	*Payload*	0.983072	0.983072	0.983072



Grafik 4.10. Tingkat Akurasi Tiap Algoritma dalam Mendeteksi Serangan RCE



Grafik 4.11. ToP Tiap Algoritma dalam Mendeteksi Serangan XSS

4.1.2. Metode Mitigasi Menggunakan *Multi-Layer Security*

Untuk menguji efektivitas *multi-layer security*, dua aplikasi simulasi serangan digunakan, yaitu Arachni dan ZAP. Kedua aplikasi tersebut digunakan untuk melakukan serangan XSS, SQLi, dan RCE. Serangan dilakukan melalui beberapa tahapan, yaitu *unimplemented* (belum menerapkan *multi-layer security*), lapisan pertama atau *OWASP ModSecuriy,* lapisan kedua atau *HTTP Middleware*, lapisan ketiga atau *Template Engine*, lapisan keempat atau *Data Sanitizer*, dan lapisan kelima atau *Framework/CMS Built-in Security*. Berikut ini adalah rincian hasil imlementasi *multi-layer security* untuk memitigasi serangan XSS, SQLi, dan RCE.

4.1.2.1. Teknik Serangan *Cross Site Scripting* (XSS)

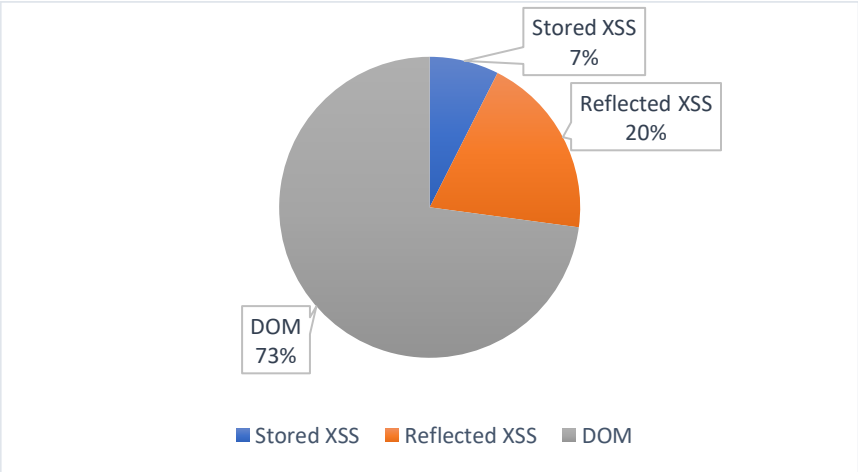
Baik Arachni maupun ZAP memiliki fitur atau plugin untuk melakukan serangan XSS. Kedua aplikasi ini mampu melakukan simulasi serangan pada 3 jenis serangan XSS, yaitu *reflected*, *stored,* dan DOM. Aplikasi yang diujicobakan pada penelitian telah didesain agar memiliki ketiga jenis celah keamanan XSS tersebut. Dengan de-mikian, efektivitas tiap lapisan atau *layer* dapat diobservasi secara mudah. Selain itu, pada lapisan kedua, yang merupakan lapisan yang berhubungan langsung dengan aplikasi, dilakukan pengujian secara *standalone* untuk melihat tingkat efektivitasnya apabila diserang tanpa lapisan pertama. Hasil implementasi metode *multi-layer security* pada serangan XSS dapat dilihat pada berikut.

Tabel 4.13. Hasil Ujicoba Serangan XSS Menggunakan Arachni

	Tahapan Penyerangan	**Jenis Serangan XSS**			**Total**
		Stored	***Reflected***	***DOM***	
1	*Unimplemented*	14	37	137	188
2	*Layer 1: OWASP ModSecurity*	14	0	0	14

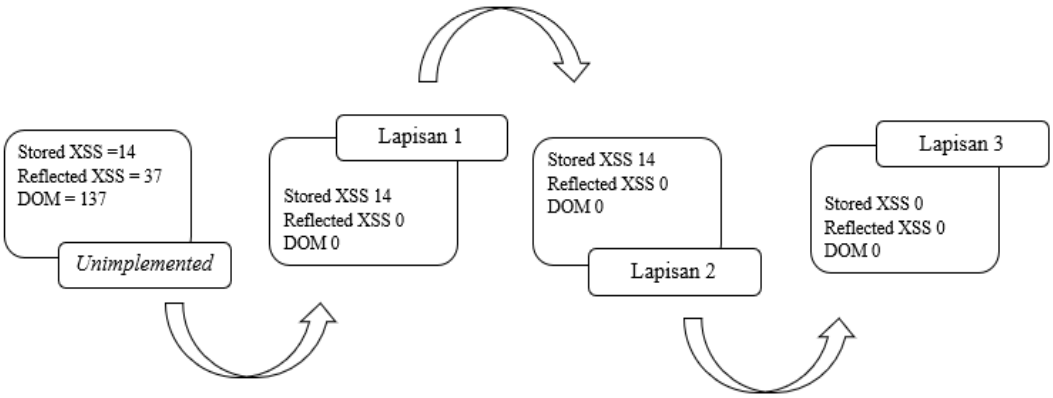
	Tahapan Penyerangan	Jenis Serangan XSS			Total
		Stored	***Reflected***	***DOM***	
3	*Layer 2: HTTP Middleware*	14	0	0	14
4	*Layer 3: Template Engine*	0	0	0	0
5	*Layer 4: Data Sanitizer*	0	0	0	0
6	*Layer 5: Framework/CMS Built-in Security*	0	0	0	0

Berdasarkan tabel 4.13 tersebut, terdapat 188 serangan yang dapat dilakukan pada *website* yang belum menerapkan *multi-layer security*. Rincian serangan yang dapat dilakukan pada tahapan ini adalah serangan *stored* XSS sebanyak 14 serangan, *reflected* XSS sebanyak 37 serangan, dan DOM XSS sebanyak 137 serangan. Serangan XSS merupakan serangan siber kategori *high severity* atau serangan dengan tingkat keparahan tinggi. Artinya, keberadaan celah keamanan XSS merupakan sesuatu yang membahayakan, meskipun jumlahnya sedikit. Dengan celah keamanan XSS tersebut, serangan metode lanjutan dapat dilakukan apabila celah keamanan XSS tidak ditangani. Contoh metode serangan lanjutan berbahaya yang dapat dilakukan setelah mengeksploitasi celah keamanan XSS adalah *account takeover*. Persentase jenis serangan XSS dapat dilihat pada gambar 4.6.



Gambar 4.10. Persentase Serangan XSS pada Tahapan *Unimplemented*

Implementasi *multi-layer security* dilakukan secara bertahap. Setelah lapisan pertama diimplementasikan, total serangan XSS yang dapat dilakukan menjadi 14 serangan dengan jenis *stored XSS.* Untuk menangani serangan *stored* XSS, lapisan kedua diimplementasikan. Namun, lapisan kedua tetap tidak berhasil memitigasi serangan tersebut. Oleh karena itu, lapisan ketiga kemudian dimplementasikan dan lapisan ini berhasil memitigasi serangan *stored* XSS. Dengan demikian, serangan XSS dapat ditangani hanya dengan tiga lapisan mitigasi (gambar 4.11).



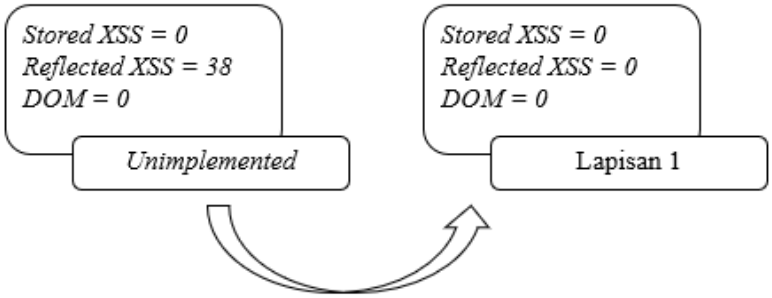
Gambar 4.11. Tahapan Mitigasi Serangan XSS Menggunakan Arachni

Selain menggunakan Arachni, serangan XSS juga menggunakan ZAP. Hasil uji coba serangan XSS menggunakan ZAP dapat dilihat pada tabel 4.14 berikut ini.

Tabel 4.14. Hasil Ujicoba Serangan XSS Menggunakan ZAP

	Tahapan Penyerangan	**Jenis Serangan XSS**			**Total**
		Stored	***Reflected***	***DOM***	
1	*Unimplemented*	0	38	0	38
2	*Layer 1: OWASP ModSecurity*	0	0	0	0
3	*Layer 2: HTTP Middleware*	0	0	0	0
4	*Layer 3: Template Engine*	0	0	0	0
5	*Layer 4: Data Sanitizer*	0	0	0	0
6	*Layer 5: Framework/CMS Built-in Security*	0	0	0	0

Sementara itu, seperti yang tertera pada tabel 4.14, total serangan XSS yang dapat dilakukan oleh ZAP sebanyak 38 serangan, yang semuanya merupakan jenis serangan *reflected* XSS. Dengan kata lain, ZAP tidak berhasil melakukan serangan jenis *stored* dan DOM XSS. Berdasarkan penelitian yang telah dilakukan, serangan XSS oleh ZAP berhasil dimitigasi hanya oleh lapisan pertama.



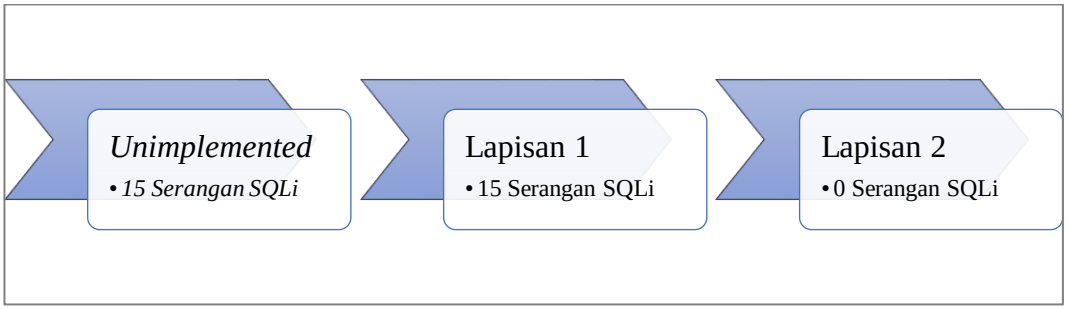
Gambar 4.12. Tahapan Mitigasi Serangan XSS Menggunakan ZAP

4.1.2.2. Teknik Serangan *SQL Injection* (SQLi)

Serangan kedua yang dilakukan adalah SQLi. Berdasarkan eksperimen yang telah dilakukan, jumlah serangan yang dapat dilakukan oleh Arachni dan ZAP sebanyak 15 serangan SQLi. Rincian serangan SQLi yang dapat dilakukan Arachni dan ZAP dapat dilihat pada tabel 4.10 dan 4.11.

Tabel 4.15 Hasil Ujicoba Serangan SQLi Menggunakan Arachni

	Tahapan Penyerangan	***SQL Injection***
1	*Unimplemented*	15
2	*Layer 1: OWASP ModSecurity*	15
3	*Layer 2: HTTP Middleware*	0
4	*Layer 3: Template Engine*	0
5	*Layer 4: Data Sanitizer*	0
6	*Layer 5: Framework/CMS Built-in Security*	0



Gambar 4.13. Tahapan Mitigasi Serangan SQLi Menggunakan Arachni

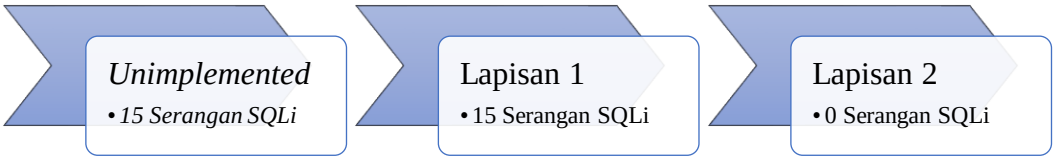
Seperti yang telah tertera pada tabel 4.15, serangan SQLi oleh ZAP dapat dimitigasi oleh lapisan kedua. Sedikit berbeda dengan serangan yang dilakukan Arachni, sera-ngan SQLi yang dilakukan ZAP dapat dimitigasi hanya oleh lapisan pertama. Pada saat pelaksanaan penelitian berlangsung, Arachni dan ZAP memiliki cara yang berbeda dalam melakukan serangan. Arachni melakukan serangan lebih *deep*, *thread* yang lebih banyak, dan kecepatan yang lebih tinggi. Dengan metode serangan tersebut, Arachni melakukan *crawling* lebih banyak dibandingkan ZAP. Oleh karena itu, Arachni dapat menembus lapisan pertama, sementara ZAP tidak.

Tabel 4.16. Hasil Ujicoba Serangan SQLi Menggunakan ZAP

	Tahapan Penyerangan	***SQL Injection***
1	*Unimplemented*	15
2	*Layer 1: OWASP ModSecurity*	0
3	*Layer 2: HTTP Middleware*	0
4	*Layer 3: Template Engine*	0
5	*Layer 4: Data Sanitizer*	0
6	*Layer 5: Framework/CMS Built-in Security*	0

ZAP dan lapisan pertama atau OWASP ModSecurity adalah produk *open source* dan sama-sama dikembangkan OWASP. ZAP adalah aplikasi untuk mengu-ji tingkat keamanan sistem dan OWASP ModSecurity untuk meningkatkan perlin-dungan aplikasi web Dalam pengembangannya, keduanya saling melengkapi, bersi-

nergi dalam meningkatkan keamanan sistem. Celah keamanan yang ditemukan dan berbagai teknik yang digunakan ZAP menjadi bahan pendukung pengembangan OWASP ModSecurity. Oleh karena itu, ZAP tidak dapat menembus pertahanan keamanan atau *Core Rule Set* yang diterapkan OWASP ModSecurity.



Gambar 4.14. Tahapan Mitigasi Serangan SQLi Menggunakan ZAP

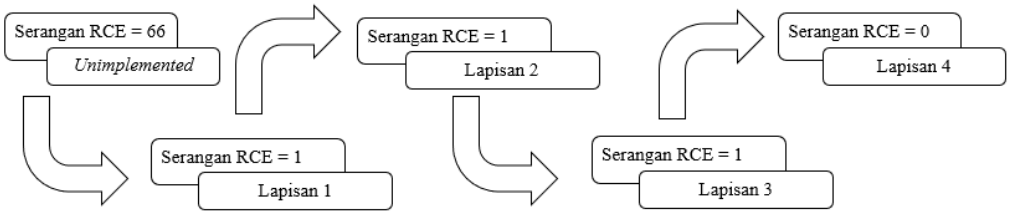
4.1.2.3. Teknik Serangan *Remote Code Execution* (RCE)

Berdasarkan eksperimen terhadap serangan RCE, *multi-layer security* berhasil mengatasi serangan RCE yang dilakukan oleh Arachni dan ZAP. Senada dengan serangan sebelumnya, jumlah serangan yang mampu dilakukan Arachni jauh lebih banyak dari ZAP. Arachni berhasil melakukan 66 serangan RCE pada tahapan *unimplemented*, sementara ZAP hanya berhasil melakukan 10 serangan RCE pada tahapan tersebut. Bahkan, serangan RCE oleh Arachni mampu menembus tiga lapisan keamanan *multi-layer security*. Rincian jenis dan total serangan RCE dapat dilihat pada tabel 4.12 dan 4.13 berikut ini.

Tabel 4.17. Hasil Ujicoba Serangan RCE Menggunakan Arachni

	Tahapan Penyerangan	***Remote Code Execution***
1	*Unimplemented*	66
2	*Layer 1: OWASP ModSecurity*	1
3	*Layer 2: HTTP Middleware*	1
4	*Layer 3: Template Engine*	1
5	*Layer 4: Data Sanitizer*	0
6	*Layer 5: Framework/CMS Built-in Security*	0

Dari 66 serangan yang berhasil dilakukan pada tahap *unimplemented*, terdapat 65 serangan yang mampu dimitigasi. Namun demikian, satu *payload* serangan RCE sukses menembus tiga lapisan keamanan pada *multi-layer security*. Serangan RCE tersebut kemudian berhasil digagalkan oleh lapisan keempat, yaitu *data sanitizer*. Tahapan mitigasi serangan RCE dapat dilihat pada gambar 4.15 di bawah ini.



Gambar 4.15. Tahapan Mitigasi Serangan RCE Menggunakan Arachni

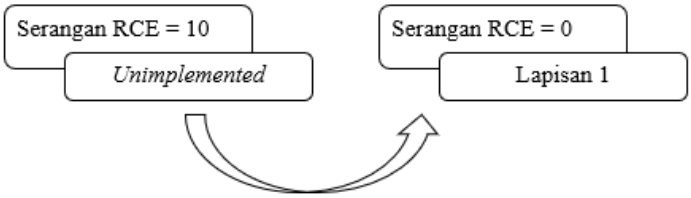
Pola atau tahapan mitigasi serangan RCE menggunakan ZAP serupa dengan pola mitigasi SQLi. Baik serangan SQLi dan RCE oleh ZAP sama-sama dapat dimitigasi hanya oleh lapisan pertama. Artinya, *multi-layer security* lapisan 2 – 5 tidak perlu bekerja atau melakukan mitigasi serangan. Terkait dengan hal tersebut, hasil ujicoba serangan RCE oleh ZAP dapat dilihat pada tabel 4.13 berikut ini.

Tabel 4.18. Hasil Ujicoba Serangan RCE Menggunakan ZAP

	Tahapan Penyerangan	***Remote Code Execution***
1	*Unimplemented*	10
2	*Layer 1: OWASP ModSecurity*	0
3	*Layer 2: HTTP Middleware*	0
4	*Layer 3: Template Engine*	0
5	*Layer 4: Data Sanitizer*	0
6	*Layer 5: Framework/CMS Built-in Security*	0

Berdasarkan tabel 4.18, serangan RCE yang dapat dilakukan pada tahap *unimplemented* adalah 10 serangan. Setelah *multi-layer security* diimplementasikan, sera-

ngan RCE oleh ZAP berhasil dimitigasi hanya oleh lapisan pertama. Visualisasi tahapan mitigasi serangan RCE dapat dilihat pada gambar 4.16 berikut.



Gambar 4.16. Tahapan Mitigasi Serangan RCE Menggunakan ZAP

4.2. Analisa Pembahasan

Eksperimen atau riset dilaksanakan dalam durasi waktu tiga bulan, sesuai dengan jadwal penelitian yang telah diungkapkan pada sub bab 3.5. Bulan pertama, penelitian difokuskan pada eksperimen metode deteksi menggunakan *machine learning*. Pada bulan kedua, eksperimen difokuskan pada metode mitigasi menggunakan *multi layer security*. Lalu, pada bulan ketiga, eksperimen difokuskan upaya mengoptimalisasikan metode deteksi dan mitigasi. Dengan optimalisasi tersebut, hasil penelitian diharapkan dapat lebih representatif dan komprehensif. Berikut ini disampaikan hasil penelitian yang dibagi menjadi dua bagian utama yaitu (1) implementasi metode deteksi menggunakan *machine learning* dan (2) implementasi metode miti-gasi menggunakan *multi-layer security.*

4.2.1. Implementasi Metode Deteksi Menggunakan *Machine Learning*

Model deteksi ini menggunakan klasifikasi biner dengan dua kelas atau klasifikasi, yaitu *payload* dan *non-payload*. Kelas *payload* dapat direpresentasikan dengan angka 1 dan *non-payload* direpresentasikan dengan angkat **0**. Adapun penjelasan masing-masing kelas metode deteksi adalah sebagai berikut.

1) ***Payload* (muatan)** yang berarti bahwa *sentence* atau baris dataset tersebut

memiliki muatan kode serangan atau merupakan kode yang berbahaya;

2) ***Non-Payload*** **(tidak memuat)** yang berarti bahwa *sentence* atau baris dataset tersebut tidak memiliki kode serangan atau bukan kode yang berbahaya.

Selain menggunakan lima algoritma untuk mendapatkan metode deteksi yang akurat, eksperimen juga mengoptimalisasikan *feature vector* dengan beberapa tahapan percobaan, sehingga metode deteksi mendapatkan tingkat akurasi yang baik. Proses *data training* dan *data testing* juga dilakukan dalam 10 s.d 30 percobaan untuk mendapatkan distribusi dataset yang benar-benar representatif. Baik XSS, SQLi, maupun RCE merupakan teknik serangan dengan basis teks, sehingga elaborasi *feature vector* banyak melibatkan metode *text processing* atau pemrosesan teks. Untuk mengefek-tifkan pemrosesan teks,

Dalam hal ini, elaborasi *feature vector* dilakukan secara dinamis dengan beberapa unit konfigurasi, yaitu *margin, punctuation removing, lowercase conversion,* dan *alphanumeric filter*. Penentuan *margin* menentukan batas vektor pada masing-masing label. Penentuan *punctuation removing, lowercase conversion,* dan *alpha-numeric filter* menentukan bagaimana vektor diolah dan disanitasi oleh algoritma. Adapun penjelasan terkait implementasi metode deteksi, optimalisasi dan evaluasi kinerja metode deteksi, dan konfigurasi parameter *feature vector* disampaikan pada penjelasan 4.2.1.1, 4.2.1.2, dan 4.2.1.3.

4.2.1.1. Teknik Serangan *Cross Site Scripting* (XSS)

Untuk menghasilkan metode yang akurat pada metode deteksi serangan XSS, terdapat dua dataset yang digunakan, yaitu (1) dataset XSS pertama dengan jumlah data 43217 [89] dan (2) dataset XSS kedua berjumlah 6606 [90].

A. *Data Preparation*

Kedua dataset serangan XSS digabungkan dan dikonversi menjadi *data frame* menggunakan pandas. Penggabungan dataset dilakukan agar mesin dapat belajar dengan data yang lebih memadai, sehingga dapat mencapai tingkat akurasi yang lebih tinggi. Dengan menggabungkan kedua dataset tersebut, data belajar dan uji juga menjadi lebih komprehensif dan representatif dengan data serangan di dunia nyata. Ketika kedua dataset digabungkan, jumlah dataset serangan XSS menjadi 49.823 baris dataset. Berikut ini adalah *sample* dengan n = 5 yang diambil dari dataset XSS pertama (tabel 4.20) dan dataset XSS kedua (tabel 4.21).

Tabel 4.19 Sample Dataset XSS Pertama

	Sentence	***Label***
606	http://www.vitelity.com/index.php?p=intl&c...	1
10778	http://localhost:8080/tienda1/publico/autentic...	0
17120	http://www.wikihow.com/create-a-new-soda&t=139...	0
7452	http://www.fin.gc.ca/finsearch/finresults_f.as...	1
17895	http://www.wikihow.com/choose-a-garage-foundat...	0

Tabel 4.20. Sample Dataset XSS Kedua

	Sentence	***Label***
4671	http://www.tb-rauxel.de/bilder/index.php?c=&qu...	1
14114	http://localhost:8080/tienda1/publico/pagar.js...	0
23727	Choosing to stay at Tortuga Bay.....	0
28652	http://www.avianca.com/inicio/navegacion/compr...	1
9760	http://cbsncis.wetpaint.com/account/%3e%3cscri...	1

B. *Data Preprocessing and Modeling*

Pemrosesan data yang telah dilakukan pada dataset XSS adalah sebagai berikut: (1) melakukan eliminasi atau menghapus baris dataset yang berstatus NaN atau kosong; (2) menghilangkan baris dataset yang duplikat pada kolom *sentence*; dan (3) melakukan konversi data pada kolom *label* menjadi data numerik untuk mempermudah proses komputasi. Dari proses tersebut, total baris dataset yang *eligible* untuk digunakan adalah 49.226, yang artinya terdapat 597 data yang tidak digunakan dari total 49.823 baris dataset.

Sementara itu, pada tahapan *data modeling*, terdapat tiga tahap yang dilakukan yaitu (1) mengubah dataset menjadi corpus karena pemrosesan data menggunakan bantuan Python NLTK; (2) mengelompokan data corpus dengan rincian 21.158 data *payload* dan 28.068 *non-payload*; (3) menyiapkan kontruksi fitur dan corpus untuk mengekstrak fitur vektor; (4) menentukan batas vektor menggunakan rumus vektor; (5) mengumpulkan fitur vektor sesuai batas ditentukan; (6) melakukan visualisasi vektor dengan *vector limit*; dan (7) membangun *feature set* menggunakan metode *bag of words* dengan bantuan library Python NLTK.

$$vector\ limit = \frac{\max(word\ counter)}{count\ of\ word\ counter}\ x\ margin$$

Rumus 4.1 Menentukan Batas Vektor

Setelah diaplikasikan rumus penentuan batas vektor, jumlah batas vektor yang didapat adalah 505 vektor. Dengan jumlah vektor tersebut, maka terdapat 505 vektor yang menjadi *identifier* label *payload* dan *non-payload*. Dengan kata lain, prediksi atau deteksi serangan XSS ditentukan berdasarkan eksistensi vektor tersebut. Berikut ini adalah daftar 100 dari 505 vektor serangan XSS, yang digunakan

untuk mendeteksi label *payload* dan *non-payload*, disertai jumlah kemunculan.

Tabel 4.21. Daftar 100 dari 505 Vektor Serangan XSS Label *Payload*

No	Vektor	Jumlah Kemunculan
1	%	99278
2	.	58055
3	;	40825
4	&	35125
5	/	34784
6	=	26954
7	gt	23021
8	3e	21640
9	>	18182
10	lt	16954
11	amp	16526
12	<	16300
13	script	16217
14	;&	16124
15	://	15715
16	(	15584
17	http	15110
18	alert	13801
19	?	12513
20	br	12427
21	1	11427
22	3c	11103
23	,	10503
24	www	8868
25	quot	8862
26	com	8635
27	22	8632
28	;/	8273

No	Vektor	Jumlah Kemunculan
29	-	7237
30	="	7074
31	=&	7013
32	3cscript	6000
33	=%	5953
34	php	5556
35	3ealert	4976
36	</	4163
37	:	4001
38	x	3983
39	20	3680
40	h1	3633
41	test	3579
42	style	3570
43	cookie	3519
44	document	3401
45	id	3365
46	31	3311
47	+	3307
48	29	3293
49	search	3256
50	asp	2903
51	marquee	2893
52	0	2634
53	27	2477
54	><	2452
55	)&	2440
56	)%	2354
57	2c	2316
58	"	2314
59	)">	2184

No	Vektor	Jumlah Kemunculan
60	xss	2029
61	contenteditable	1828
62	index	1785
63	(&	1705
64	;)&	1650
65	{	1599
66	83	1563
67	2fscript	1514
68	2f	1507
69	28	1484
70	72	1420
71	a	1407
72	('	1405
73	)></	1403
74	gov	1354
75	,-	1342
76	src	1301
77	org	1301
78	)"	1286
79	t	1283
80	by	1281
81	TRUE	1273
82	3ch1	1245
83	)	1182
84	2	1146
85	div	1100
86	tabindex	1100
87	y	1098
88	net	1081
89	>%	1031
90	q	1006

No	Vektor	Jumlah Kemunculan
91	cgi	1001
92	html	994
93	draggable	988
94	74	987
95	fromcharcode	983
96	3cmarquee	977
97	(%	960
98	/?	958
99	3	957
100	20by	955

Tabel di atas menunjukan daftar vektor yang digunakan sebagai *identifier* pada dataset serangan XSS dengan label *payload*. Seperti yang terlihat pada tabel tersebut, vektor diekstrak dengan aturan *corpus* yang menerima tanda baca dan *alphanumeric* karakter. Berdasarkan penelitian yang telah dilakukan pada metode deteksi serangan XSS, eksistensi tanda baca dan karakter *alphanumeric* dapat meningkatkan tingkat akurasi metode deteksi. Fakta ini juga sekaligus menunjukan bahwa baris kode serangan XSS dapat terdiri dari tanda baca dan karakter *alphanumeric*. Selain vektor *payload*, vektor serangan XSS dengan label *non-payload* juga dihimpun ke dalam seperangkat fitur. Adapun daftar 100 dari 505 vektor serangan XSS dengan label *non-payload* dapat dilihat pada tabel berikut.

Tabel 4.22. Daftar 100 dari 505 Vektor Serangan XSS Label *Non-Payload*

No	Vektor	Jumlah Kemunculan
1	.	219576
2	=	130807
3	t	115031
4	the	105107

No	Vektor	Jumlah Kemunculan
5	\	97530
6	&	87249
7	,	84845
8	-	68943
9	(	62535
10	a	51306
11	and	50519
12	to	43662
13	:	36622
14	/	34175
15	+	32057
16	this	29905
17	was	29028
18	i	28392
19	in	26748
20	we	25267
21	of	23893
22	{	22878
23	)	21407
24	for	20950
25	\'	19612
26	is	19495
27	it	18520
28	r	18118
29	%	17570
30	function	16081
31	hotel	15756
32	://	14920
33	http	14801
34	'	14665
35	if	14444

No	Vektor	Jumlah Kemunculan
36	\"	14300
37	on	14270
38	;	14197
39	at	14020
40	*	12832
41	room	12663
42	}	12649
43	that	12647
44	were	12462
45	with	12314
46	you	12169
47	//	11934
48	but	11056
49	var	11047
50	not	10902
51	com	10529
52	);	10063
53	n	9666
54	very	9591
55	had	9092
56	k	9057
57	our	8938
58	www	8897
59	b1	8843
60	\',	8720
61	wikihow	8578
62	mainentity	8571
63	[	8501
64	0	8426
65	there	8271
66	?	7920

No	Vektor	Jumlah Kemunculan
67	are	7842
68	from	7827
69	1	7734
70	be	7725
71	have	7703
72	(\'	7693
73	as	7635
74	return	7410
75	s	7317
76	they	7245
77	my	6921
78	\",	6819
79	great	6484
80	{\	6478
81	!	6426
82	all	6234
83	login	6082
84	so	5944
85	;\	5832
86	2	5658
87	stay	5609
88	()	5569
89	get	5478
90	an	5466
91	);\	5352
92	one	5145
93	good	5024
94	registro	5001
95	or	4915
96	localhost	4867
97	}\	4866

No	Vektor	Jumlah Kemunculan
98	would	4844
99	8080	4833
100	nombre	4826

Berdasarkan data dari kedua tabel di atas, terdapat beberapa vektor yang saling beririsan. Terdapat vektor label *payload* yang juga menjadi vektor pada label *non-payload*. Oleh karena itu, metode deteksi serangan juga mengukur derajat atau kecenderungan label berdasarkan jumlah kemunculan vektor, baik pada label *payload* maupun *non-payload*. Derajat atau kecenderungan suatu label dihitung berdasarkan jumlah kemunculan vektor pada label *payload* yang dikurangi kemunculan vektor pada label *non-payload*. Jika nilai kecenderungan positif, vektor berada pada label *payload*. Namun jika negatif, vektor berada pada label *non-payload*. Daftar vektor yang saling beririsan serta derajat dan kecenderungan posisi label vektor tersebut dapat dilihat pada tabel di berikut ini.

Tabel 4.23. Irisan Vektor Label *Payload* dan *Non-Payload* XSS

No	Vektor	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
1	%	99278	17570	81708	payload
2	.	58055	219576	-161521	non-payload
3	;	40825	14197	26628	payload
4	&	35125	87249	-52124	non-payload
5	/	34784	34175	609	payload
6	=	26954	130807	-103853	non-payload
7	>	18182	3313	14869	payload
8	<	16300	3232	13068	payload
9	script	16217	1075	15142	payload
10	://	15715	14920	795	payload

No	Vektor	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
11	(	15584	62535	-46951	non-payload
12	http	15110	14801	309	payload
13	?	12513	7920	4593	payload
14	1	11427	7734	3693	payload
15	,	10503	84845	-74342	non-payload
16	www	8868	8897	-29	non-payload
17	com	8635	10529	-1894	non-payload
18	-	7237	68943	-61706	non-payload
19	="	7074	1085	5989	payload
20	</	4163	1161	3002	payload
21	:	4001	36622	-32621	non-payload
22	x	3983	2489	1494	payload
23	test	3579	1707	1872	payload
24	style	3570	1749	1821	payload
25	document	3401	1673	1728	payload
26	id	3365	3298	67	payload
27	+	3307	32057	-28750	non-payload
28	0	2634	8426	-5792	non-payload
29	2c	2316	4295	-1979	non-payload
30	"	2314	1084	1230	payload
31	{	1599	22878	-21279	non-payload
32	2f	1507	998	509	payload
33	a	1407	51306	-49899	non-payload
34	t	1283	115031	-113748	non-payload
35	by	1281	4082	-2801	non-payload
36	TRUE	1273	4040	-2767	non-payload
37	)	1182	21407	-20225	non-payload
38	2	1146	5658	-4512	non-payload
39	div	1100	1281	-181	non-payload
40	y	1098	1644	-546	non-payload
41	html	994	1306	-312	non-payload

No	Vektor	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
42	3	957	3542	-2585	non-payload
43	title	955	877	78	payload
44	s	917	7317	-6400	non-payload
45	js	825	1582	-757	non-payload
46	var	823	11047	-10224	non-payload
47	url	789	1154	-365	non-payload
48	string	780	2003	-1223	non-payload
49	h	777	880	-103	non-payload
50	b	760	1923	-1163	non-payload
51	jsp	739	4819	-4080	non-payload
52	30	733	801	-68	non-payload
53	de	719	2062	-1343	non-payload
54	c	701	2746	-2045	non-payload
55	e	697	2305	-1608	non-payload
56	name	652	3488	-2836	non-payload
57	*	635	12832	-12197	non-payload
58	d	633	2136	-1503	non-payload
59	);	627	10063	-9436	non-payload
60	5	578	2717	-2139	non-payload
61	if	570	14444	-13874	non-payload
62	10	559	1924	-1365	non-payload
63	[	557	8501	-7944	non-payload
64	i	548	28392	-27844	non-payload
65	function	541	16081	-15540	non-payload
66	p	536	1759	-1223	non-payload
67	type	514	2418	-1904	non-payload
68	login	501	6082	-5581	non-payload
69	){	475	2416	-1941	non-payload
70	}	460	12649	-12189	non-payload
71	body	386	886	-500	non-payload
72	n	379	9666	-9287	non-payload

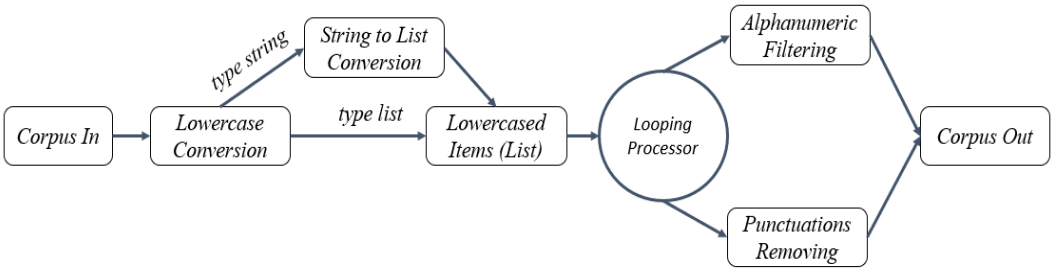
No	Vektor	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
73	here	347	2816	-2469	non-payload
74	m	343	1587	-1244	non-payload
75	left	325	1227	-902	non-payload
76	8	315	1017	-702	non-payload
77	us	315	4246	-3931	non-payload
78	me	314	2380	-2066	non-payload
79	in	313	26748	-26435	non-payload
80	it	312	18520	-18208	non-payload
81	'	310	14665	-14355	non-payload
82	data	298	2549	-2251	non-payload
83	go	295	1976	-1681	non-payload
84	r	293	18118	-17825	non-payload
85	be	293	7725	-7432	non-payload
86	4	290	2564	-2274	non-payload
87	main	289	854	-565	non-payload
88	\	285	97530	-97245	non-payload
89	@	283	4629	-4346	non-payload
90	to	279	43662	-43383	non-payload
91	l	272	1449	-1177	non-payload
92	7	264	1054	-790	non-payload
93	k	244	9057	-8813	non-payload
94	from	242	7827	-7585	non-payload
95	return	241	7410	-7169	non-payload
96	object	239	2288	-2049	non-payload
97	6	228	1270	-1042	non-payload
98	form	228	982	-754	non-payload
99	this	216	29905	-29689	non-payload
100	9	213	847	-634	non-payload
101	g	212	984	-772	non-payload
102	//	206	11934	-11728	non-payload
103	text	203	2007	-1804	non-payload

No	Vektor	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
104	new	200	4606	-4406	non-payload
105	100	197	874	-677	non-payload
106	do	195	2504	-2309	non-payload
107	code	194	1208	-1014	non-payload
108	o	190	867	-677	non-payload
109	all	188	6234	-6046	non-payload
110	email	187	4329	-4142	non-payload
111	|	171	1929	-1758	non-payload
112	FALSE	169	2875	-2706	non-payload
113	();	165	3505	-3340	non-payload
114	&&	164	2581	-2417	non-payload
115	for	152	20950	-20798	non-payload
116	out	152	4486	-4334	non-payload
117	w	151	1036	-885	non-payload
118	].	148	1758	-1610	non-payload
119	on	146	14270	-14124	non-payload
120	()	145	5569	-5424	non-payload
121	length	142	2691	-2549	non-payload
122	window	137	1290	-1153	non-payload
123	no	135	4268	-4133	non-payload
124	==	135	2113	-1978	non-payload
125	else	135	3020	-2885	non-payload
126	location	133	3886	-3753	non-payload
127	width	131	911	-780	non-payload
128	view	128	1829	-1701	non-payload

C. *Feature Enginering, Data Training, and Data Testing*

Proses *featuring enginering* pada dataset serangan XSS dibagi menjadi dua tahapan, yaitu (1) kontruksi korpus dan (2) kontruksi fitur. Pada tahapan kontruksi

kor-pus, dataset diolah atau diproses sesuai dengan *corpus rules* yang telah ditentukan. Alur kontruksi korpus serangan XSS dapat dilihat pada gambar. Sementara itu, kontruksi fitur merupakan tahapan konversi dan verifikasi eksistensi suatu vektor sebelum memasuki proses prediksi atau deteksi. Kontruksi fitur berkaitan dengan batas vektor yang telah ditentukan oleh parameter *margin*.



Gambar 4.17. Tahapan Kontruksi Corpus Datataset Serangan XSS

Kontruksi corpus di atas dieksekusi menggunakan bahasa pemrograman Python. Fungsi kontruksi corpus diatur oleh *corpus rule* atau aturan corpus yang direpresentasikan oleh variabel tipe *dictionary*. Ketika dioperasionalkan dan di-implementasikan dalam bentuk baris kode Python, baris kode tahapan atau alur kontruksi corpus di atas dapat dituliskan melaui baris kode sebagai berikut.

```
1.  def construct_corpus(corpus_in)->list:
2.      punctuations = list(string.punctuation)
3.      corpus_out   = []
4.      if type(corpus_in) == str:
5.          if corpus_rules['to_lowercase']:
6.              corpus_in = corpus_in.lower()
7.          corpus_in = word_tokenize(corpus_in)
8.      else:
9.          if corpus_rules['to_lowercase']:
10.             corpus_in = [w.lower() for w in corpus_in]
11.     for item in corpus_in:
12.         if not corpus_rules['remove_puncts']:
13.             punctuations = []
14.         if not item in punctuations:
```

```
15.             if corpus_rules['only_alphanum']:
16.                 item = re.sub(r'[\W\d]', '', item)
17.             if item:
18.                 corpus_out.append(item)
19.     return corpus_out
```

Gambar 4.18. Fungsi Kontruksi Corpus dalam Membangun Vektor

Fungsi kontruksi corpus tersebut ditempatkan dalam lingkup fungsi kontruksi fitur. Kontruksi fitur digunakan untuk memverifikasi eksistensi atau keberadaan suatu vektor. Dengan kata lain, kontruksi fitur memvalidasi apakah vektor tersebut berada pada label atau kelas *payload* atau *non-payload*. Tipe luaran fungsi ini adalah *dictionary* yang di dalamnya berisi rincian *vector* sebagai *key* dan *boolean* sebagai *value*. *Boolean True* jika baris dataset yang diprediksi memiliki anggota vektor. Adapun baris kode fungsi kontruksi fitur adalah sebagai berikut.

```
1.  def construct_features(corpus:string)->dict:
2.      feature = {}
3.      corpus  = set(construct_corpus(corpus))
4.      for vector in feature_vectors.keys():
5.          feature[vector] = vector in corpus # corpus string -> corpus list
6.      return feature
```

Gambar 4.19. Fungsi Kontruksi Fitur untuk Memverifikasi Label Vektor

Pembagian *data training* dan *data testing* pada dataset serangan XSS dilakukan dengan pengaturan 0.2 *test size*, yang artinya 80% digunakan untuk data latih dan 20% untuk data pengujian. Rasio tersebut diambil agar persentase data uji lebih representatif. Proses belajar dilakukan berulang-ulang sehingga mendapatkan tingkat akurasi tertinggi atau maksimal. Selain itu, konfigurasi parameter juga dilakukan untuk mendapatkan data *training* dan *testing* yang representatif dan komprehensif. Dengan tahapan-tahapan tersebut, tingkat akurasi metode deteksi serangan XSS semakin dapat meningkat secara bertahap.

D. *Performance Optimalization*

Serangan XSS bersifat dinamis, kompleks, dan memiliki karakteristik tersendiri. Oleh karena itu, untuk mendapatkan metode deteksi dengan tingkat akurasi tinggi, beberapa algoritma harus diujikan. Melalui pengujian lima algoritma terpilih, peneliti memiliki re-ferensi lebih komprehensif dalam menentukan algoritma terbaik. Terkait dengan hal tersebut, penelitian ini bahkan melakukan *configu-ration switcher* untuk mendapatkan kinerja metode deteksi yang paling optimal. Peneliti juga mencari konfigurasi teroptimal dalam mendeteksi serangan siber. Rekam jejak upaya optimalisasi kinerja masing-masing algoritma beserta konfigurasi parameter yang digu-nakan dapat dilihat pada tabel berikut ini. Baris tabel diurutkan berdasarkan tingkat akurasi tertinggi menuju terendah.

Tabel 4.24. Tahapan Optimalisasi Kinerja Metode Deteksi XSS Lima Algoritma

	Classifier Name	***Accuracy***	***ToP***	***Margin***	***Vector Limit***	***Total Test***	***Total Train***	***Lowercase***	***Alphanumeric***	***Remove Puncts***
1	Support Vector Machine	0,9965	0:03:46	5	505	9846	39380	True	False	False
2	K-Nearest Neigbors	0,9960	0:05:52	2	202	9846	39380	True	False	False
3	K-Nearest Neigbors	0,9959	0:05:47	2.5	252	9846	39380	True	False	False
4	Support Vector Machine	0,9958	0:01:20	2.5	252	9846	39380	True	False	False
5	Support Vector Machine	0,9958	0:00:12	0.5	50	9846	39380	True	False	False
6	K-Nearest Neigbors	0,9958	0:21:38	5	505	9846	39380	True	False	False
7	K-Nearest Neigbors	0,9956	0:15:10	7	707	9846	39380	True	False	False

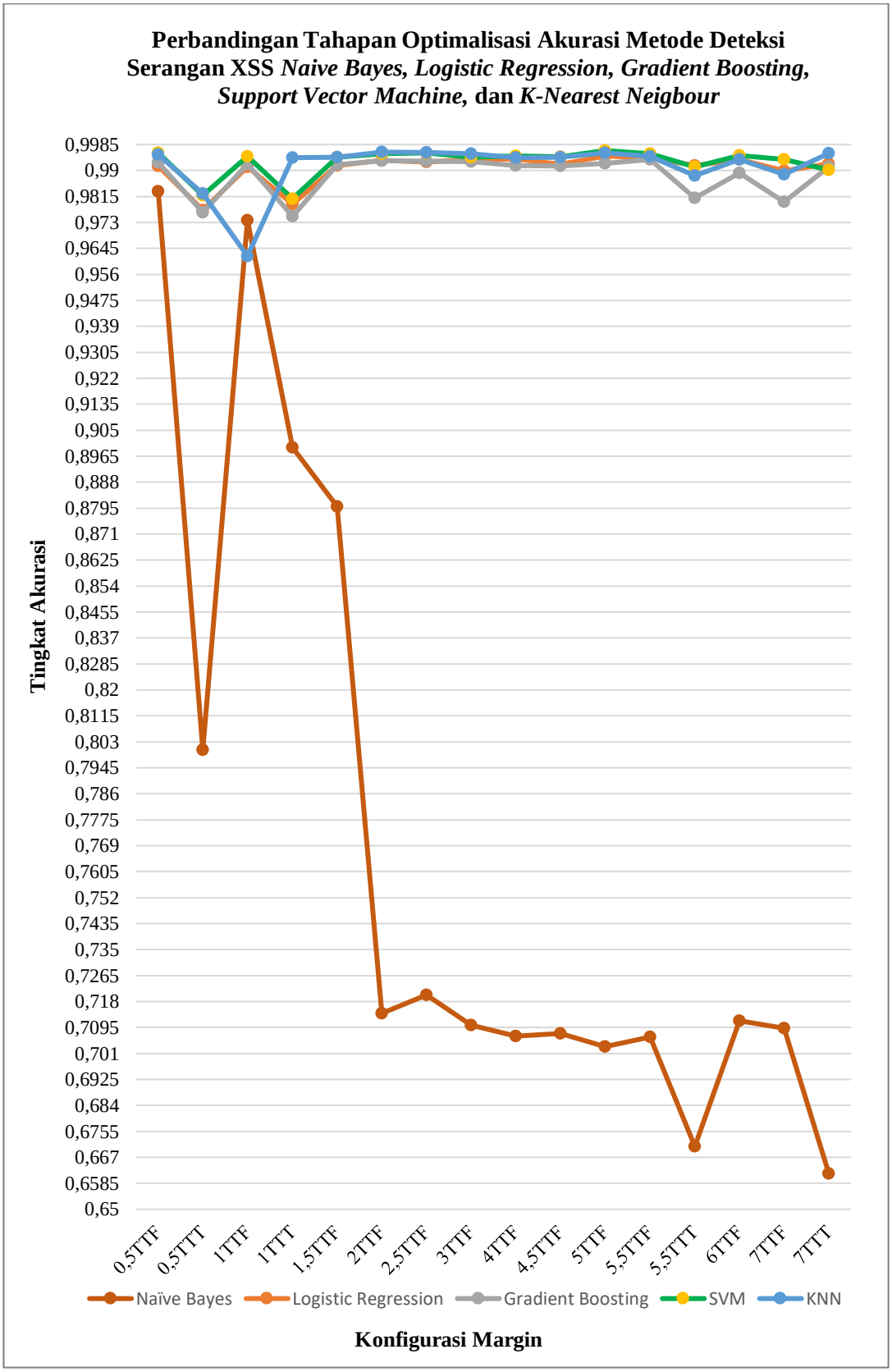
	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
8	Support Vector Machine	0,9955	0:02:47	5.5	556	9846	39380	True	False	False
9	K-Nearest Neigbors	0,9954	0:06:21	3	303	9846	39380	True	False	False
10	Support Vector Machine	0,9954	0:00:54	2	202	9846	39380	True	False	False
11	K-Nearest Neigbors	0,9952	0:01:36	0.5	50	9846	39380	True	False	False
12	Support Vector Machine	0,9949	0:04:30	6	606	9846	39380	True	False	False
13	Logistic Regression	0,9947	0:00:40	5	505	9846	39380	True	False	False
14	Support Vector Machine	0,9947	0:02:30	4	404	9846	39380	True	False	False
15	K-Nearest Neigbors	0,9946	0:12:00	5.5	556	9846	39380	True	False	False
16	Support Vector Machine	0,9946	0:00:19	1	101	9846	39380	True	False	False
17	Support Vector Machine	0,9945	0:02:57	4.5	455	9846	39380	True	False	False
18	K-Nearest Neigbors	0,9944	0:10:34	4.5	455	9846	39380	True	False	False
19	K-Nearest Neigbors	0,9944	0:03:43	1.5	151	9846	39380	True	False	False
20	Support Vector Machine	0,9943	0:00:41	1.5	151	9846	39380	True	False	False
21	Support Vector Machine	0,9943	0:01:16	3	303	9846	39380	True	False	False
22	Logistic Regression	0,9942	0:00:34	5.5	556	9846	39380	True	False	False
23	K-Nearest Neigbors	0,9942	0:02:57	1	101	9846	39380	True	False	False
24	K-Nearest Neigbors	0,9942	0:13:20	4	404	9846	39380	True	False	False
25	Logistic Regression	0,9938	0:00:27	4	404	9846	39380	True	False	False
26	Support Vector Machine	0,9937	0:03:48	7	707	9846	39380	True	False	False

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
27	Logistic Regression	0,9937	0:00:17	3	303	9846	39380	True	False	False
28	Logistic Regression	0,9937	0:00:47	6	606	9846	39380	True	False	False
29	K-Nearest Neigbors	0,9937	0:15:34	6	606	9846	39380	True	False	False
30	Gradient Boosting	0,9936	0:01:52	5.5	556	9846	39380	True	False	False
31	Logistic Regression	0,9934	0:00:13	2	202	9846	39380	True	False	False
32	Gradient Boosting	0,9932	0:00:41	2	202	9846	39380	True	False	False
33	Gradient Boosting	0,9930	0:00:53	2.5	252	9846	39380	True	False	False
34	Gradient Boosting	0,9929	0:00:57	3	303	9846	39380	True	False	False
35	Logistic Regression	0,9928	0:00:16	2.5	252	9846	39380	True	False	False
36	Gradient Boosting	0,9926	0:00:12	0.5	50	9846	39380	True	False	False
37	Gradient Boosting	0,9923	0:01:53	5	505	9846	39380	True	False	False
38	Logistic Regression	0,9922	0:00:44	7	707	9846	39380	True	False	False
39	Logistic Regression	0,9921	0:00:37	4.5	455	9846	39380	True	False	False
40	Gradient Boosting	0,9921	0:00:19	1	101	9846	39380	True	False	False
41	Gradient Boosting	0,9920	0:00:30	1.5	151	9846	39380	True	False	False
42	Logistic Regression	0,9917	0:00:34	5.5	556	9846	39380	True	True	True
43	Gradient Boosting	0,9917	0:01:38	4	404	9846	39380	True	False	False
44	Logistic Regression	0,9916	0:00:09	1.5	151	9846	39380	True	False	False
45	Gradient Boosting	0,9915	0:01:44	4.5	455	9846	39380	True	False	False

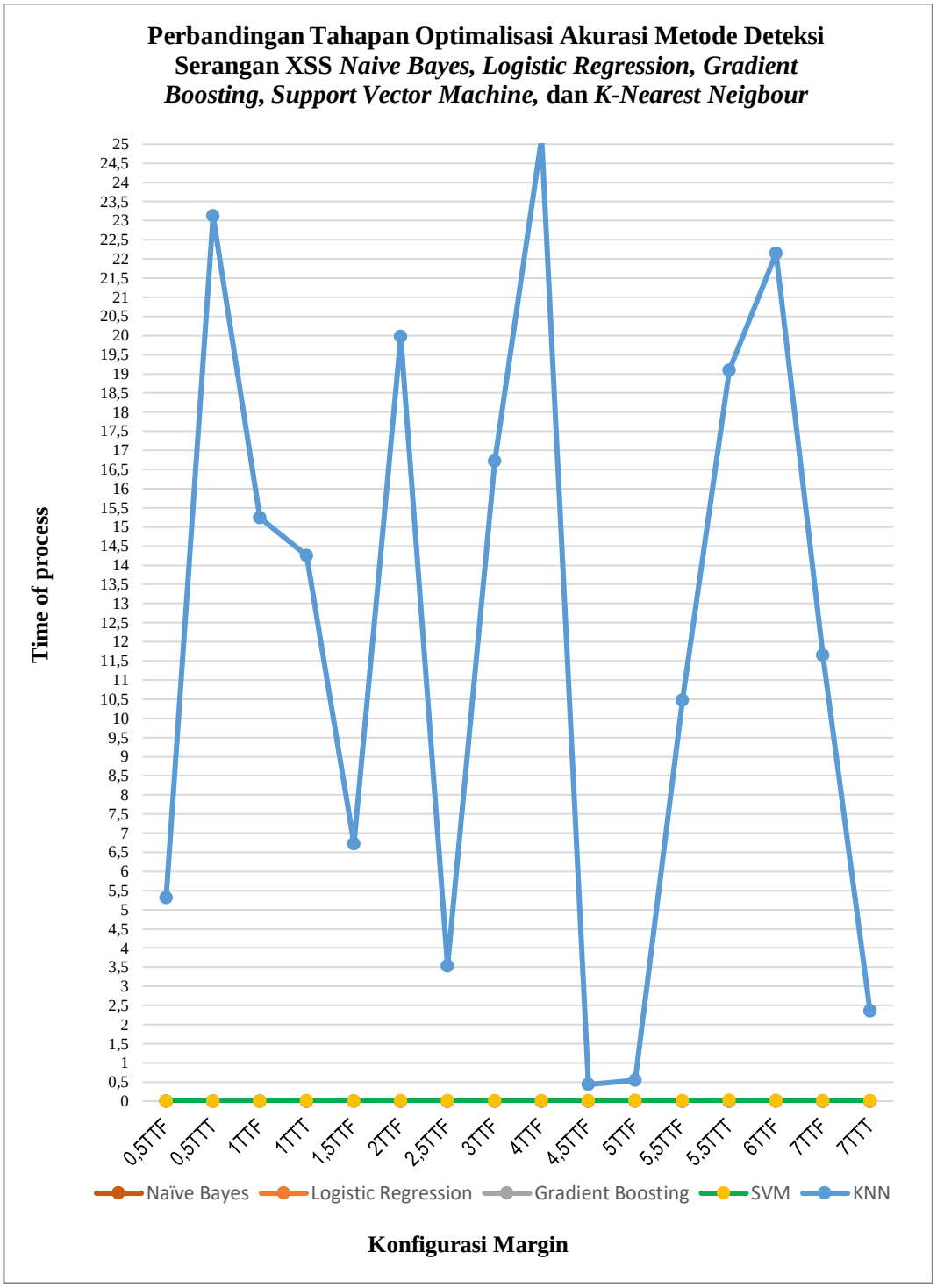
	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
46	Logistic Regression	0,9915	0:00:03	0.5	50	9846	39380	True	False	False
47	Gradient Boosting	0,9913	0:02:38	7	707	9846	39380	True	False	False
48	Support Vector Machine	0,9913	0:05:57	5.5	556	9846	39380	True	True	True
49	Logistic Regression	0,9912	0:00:06	1	101	9846	39380	True	False	False
50	Support Vector Machine	0,9902	0:05:04	7	707	9846	39380	True	True	True
51	Logistic Regression	0,9899	0:00:38	7	707	9846	39380	True	True	True
52	Gradient Boosting	0,9892	0:02:15	6	606	9846	39380	True	False	False
53	K-Nearest Neigbors	0,9887	0:14:23	7	707	9846	39380	True	True	True
54	K-Nearest Neigbors	0,9883	0:13:15	5.5	556	9846	39380	True	True	True
55	Naive Bayes	0,9832	0:00:03	0.5	50	9846	39380	True	False	False
56	K-Nearest Neigbors	0,9824	0:01:25	0.5	50	9846	39380	True	True	True
57	Support Vector Machine	0,9819	0:00:23	0.5	50	9846	39380	True	True	True
58	Gradient Boosting	0,9810	0:01:52	5.5	556	9846	39380	True	True	True
59	Support Vector Machine	0,9808	0:00:40	1	101	9846	39380	True	True	True
60	Gradient Boosting	0,9797	0:01:51	7	707	9846	39380	True	True	True
61	Logistic Regression	0,9791	0:00:07	1	101	9846	39380	True	True	True
62	Logistic Regression	0,9770	0:00:03	0.5	50	9846	39380	True	True	True
63	Gradient Boosting	0,9763	0:00:09	0.5	50	9846	39380	True	True	True
64	Gradient Boosting	0,9750	0:00:19	1	101	9846	39380	True	True	True

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
65	Naive Bayes	0,9737	0:00:05	1	101	9846	39380	True	False	False
66	K-Nearest Neigbors	0,9620	0:02:58	1	101	9846	39380	True	True	True
67	Naive Bayes	0,8993	0:00:06	1	101	9846	39380	True	True	True
68	Naive Bayes	0,8801	0:00:09	1.5	151	9846	39380	True	False	False
69	Naive Bayes	0,8004	0:00:02	0.5	50	9846	39380	True	True	True
70	Naive Bayes	0,7201	0:00:15	2.5	252	9846	39380	True	False	False
71	Naive Bayes	0,7141	0:00:14	2	202	9846	39380	True	False	False
72	Naive Bayes	0,7117	0:00:53	6	606	9846	39380	True	False	False
73	Naive Bayes	0,7102	0:00:17	3	303	9846	39380	True	False	False
74	Naive Bayes	0,7093	0:00:43	7	707	9846	39380	True	False	False
75	Naive Bayes	0,7075	0:00:33	4.5	455	9846	39380	True	False	False
76	Naive Bayes	0,7067	0:00:28	4	404	9846	39380	True	False	False
77	Naive Bayes	0,7064	0:00:32	5.5	556	9846	39380	True	False	False
78	Naive Bayes	0,7032	0:00:44	5	505	9846	39380	True	False	False
79	Naive Bayes	0,6705	0:00:36	5.5	556	9846	39380	True	True	True
80	Naive Bayes	0,6617	0:00:39	7	707	9846	39380	True	True	True

Berdasarkan tabel tersebut, SVM bersaing ketat dengan KNN. KNN mampu mendapatkan tingkat akurasi tidak terlalu jauh dari SVM, namun ToP KNN relatif lama, sehingga memengaruhi kecepatan deteksi. Berikut adalah grafik perbandingan tahapan optimalisasi.



Grafik 4.12. Perbandingan Tahapan Optimalisasi Akurasi Metode Deteksi XSS

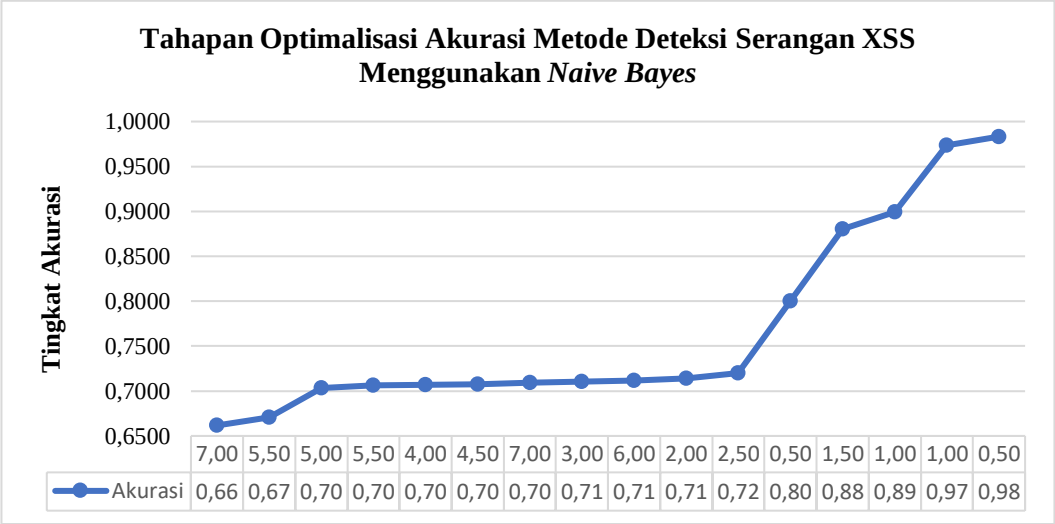


Grafik 4.13.. Perbandingan Tahapan Optimalisasi ToP Metode Deteksi XSS

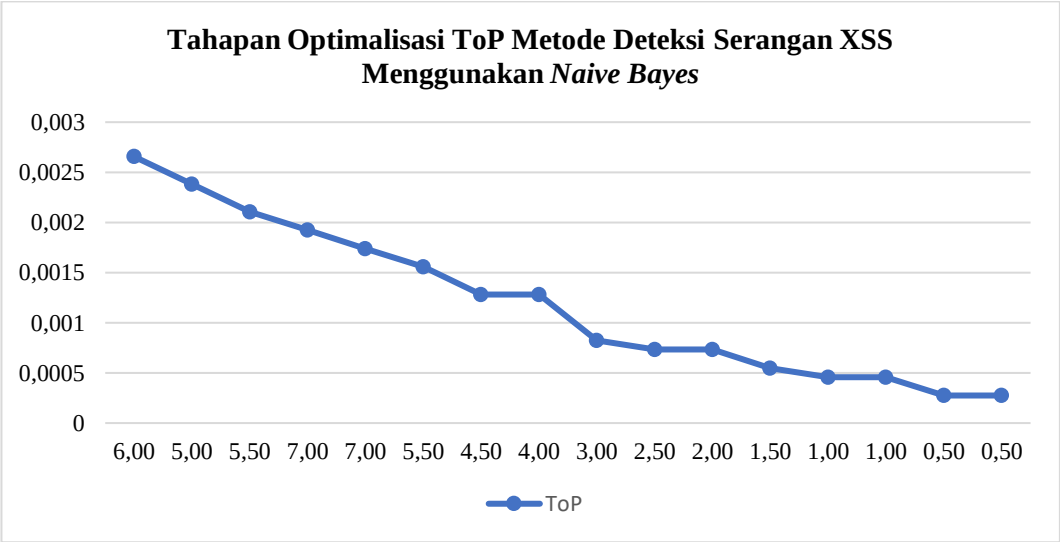
Melalui grafik perbandingan tersebut, perbedaan tingkat akurasi dan ToP pada masing-masing algoritma dapat terlihat secara representatif. Pada tahapan optimalisasi akurasi, Naïve Bayes memiliki rekam jejak tingkat akurasi yang sangat di-

namis dan signifikan. Sementara itu, pada tahapan optimalisasi ToP, KNN memiliki rekam jejak ToP yang jauh berbeda dengan algoritma yang lain. Selain grafik perbandingan tersebut, agar pembahasan penelitian ini lebih komprehensif, berikut ini ditampilkan tahapan optimalisasi akurasi dan ToP masing-masing algoritma.

1) Tingkat Akurasi dan ToP Naïve Bayes



Grafik 4.14. Tahapan Optimalisasi Akurasi Metode Deteksi XSS Naive Bayes

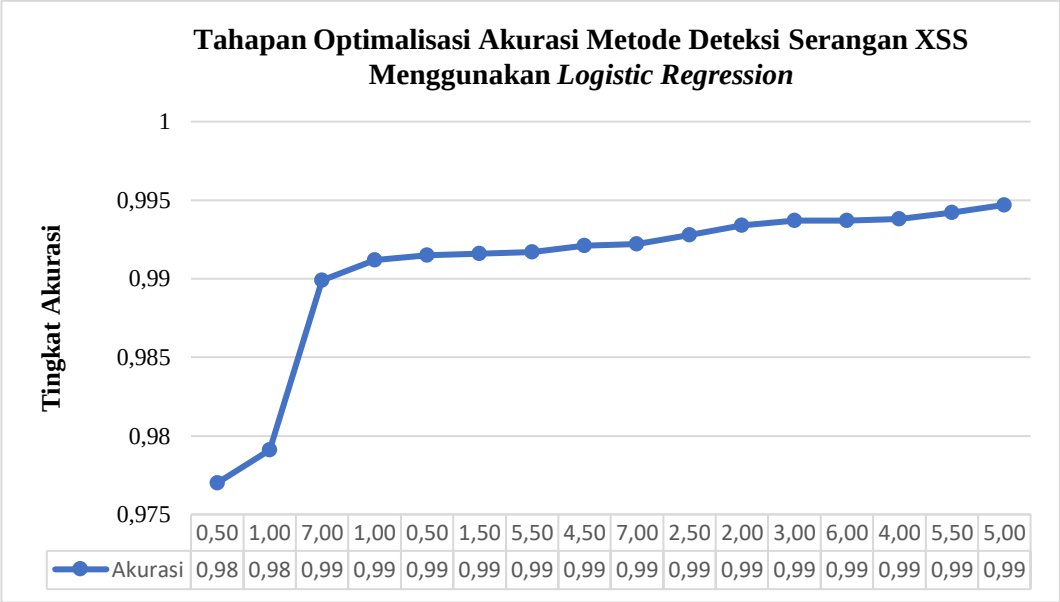


Grafik 4.15. Tahapan Optimalisasi Akurasi Metode Deteksi XSS Naive Bayes

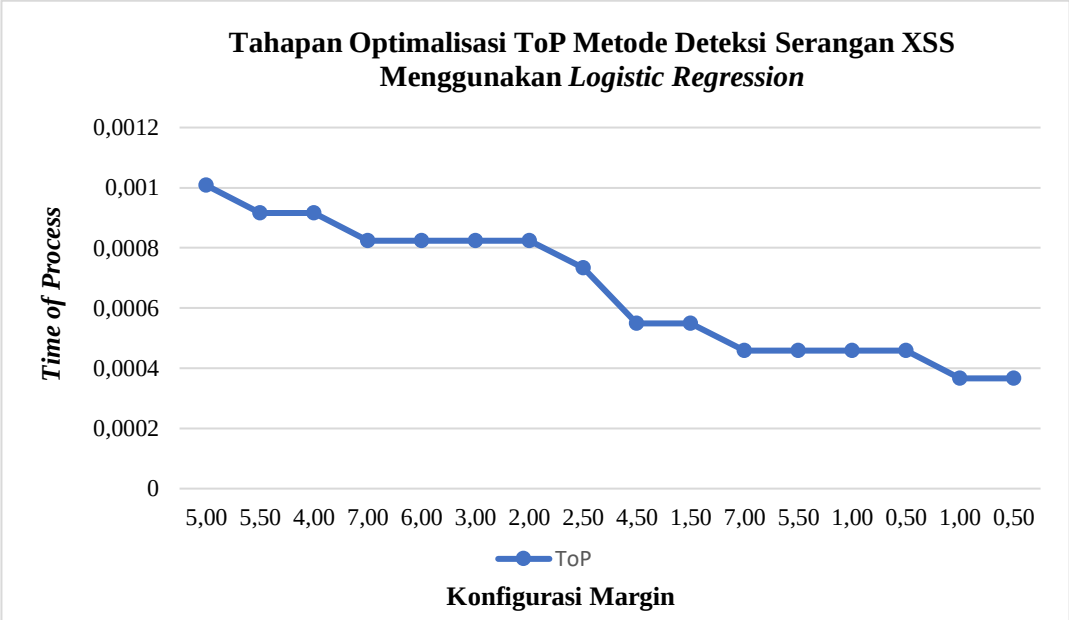
Sesuai dengan grafik 4.32 dan 4.33, Naïve Bayes mendapatkan tingkat akurasi 0,98 dengan ToP 0,00027 mikro detik dengan margin 0,5. Semakin bertambah

margin, tingkat akurasi semakin menurun dan ToP semakin meningkat. Fenomena ini sedikit berbeda dengan algoritma lainnya. Pada algoritma lain, penambahan *margin* diikuti dengan meningkatnya tingkat akurasi.

2) Tingkat Akurasi dan ToP *Logistic Regression*



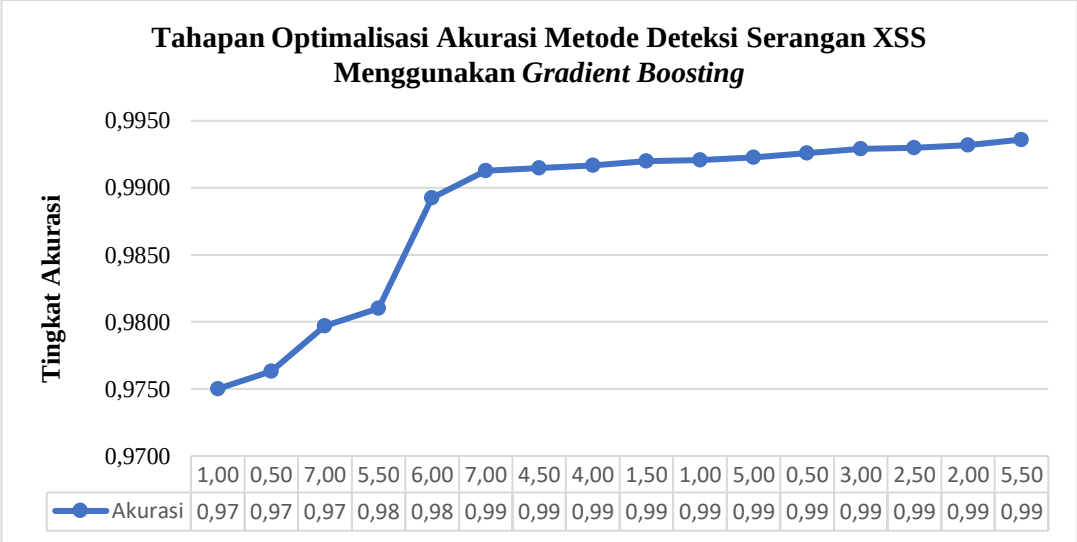
Grafik 4.16. Tahapan Optimalisasi Akurasi Metode Deteksi XSS *Logistic Regression*



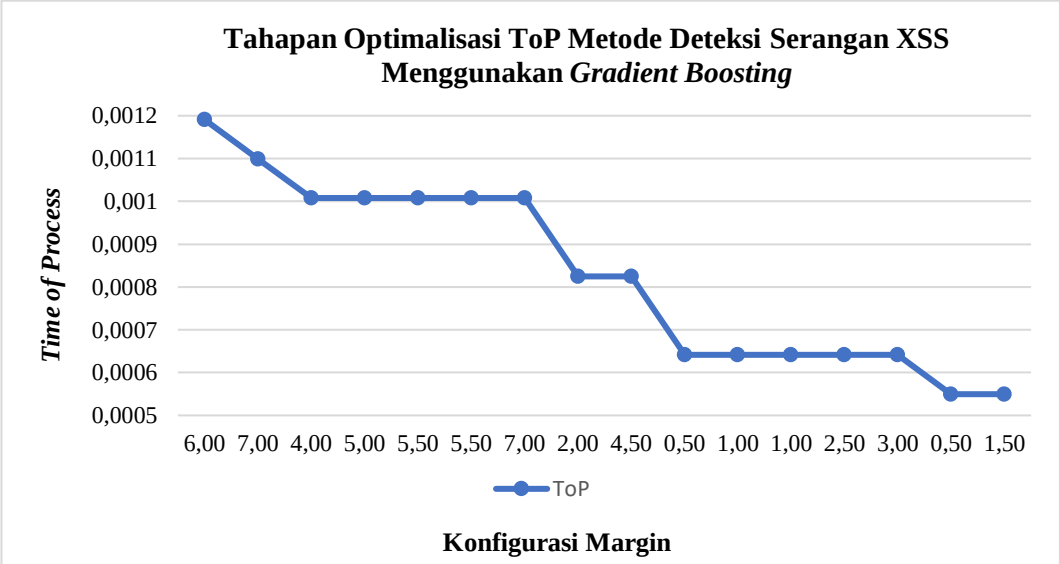
Grafik 4.17. Tahapan Optimalisasi ToP Metode Deteksi XSS *Logistic Regression*

Tingkat akurasi tertinggi *Logistic Regression* dalam mendeteksi serangan XSS mencapai 0,9947 dengan ToP 0,001 mikro detik. Baik pada optimalisasi akurasi maupun ToP, penambahan margin diikuti dengan kenaikan tingkat akurasi dan ToP. Tingkat akurasi terendah algoritma ini berada pada angka 0,977 dengan besar margin 0,5, diikuti dengan ToP terendah yang berada pada angka 0,00036.

3) Tingkat Akurasi dan ToP *Gradient Boosting*



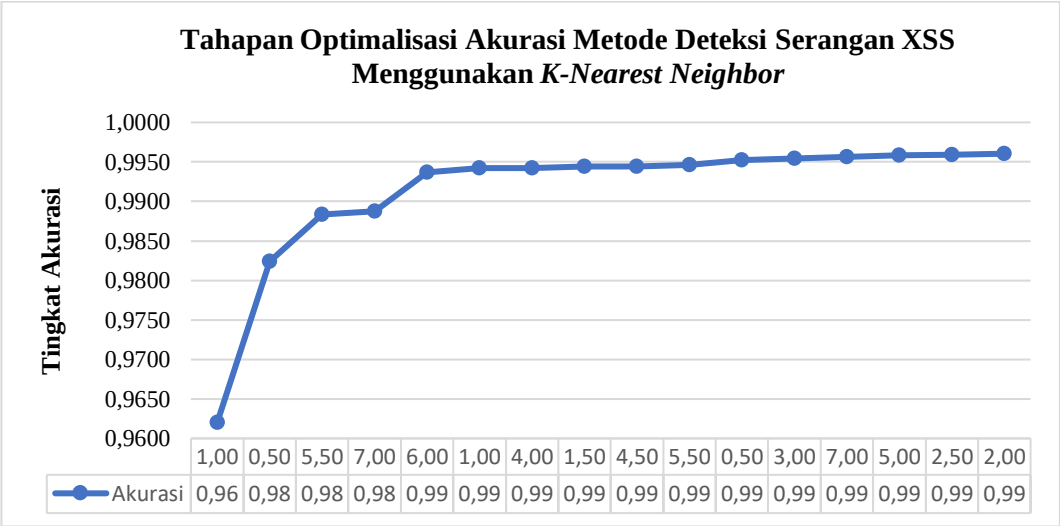
Grafik 4.18. Tahapan Optimalisasi Akurasi Metode Deteksi XSS *Gradient Boosting*



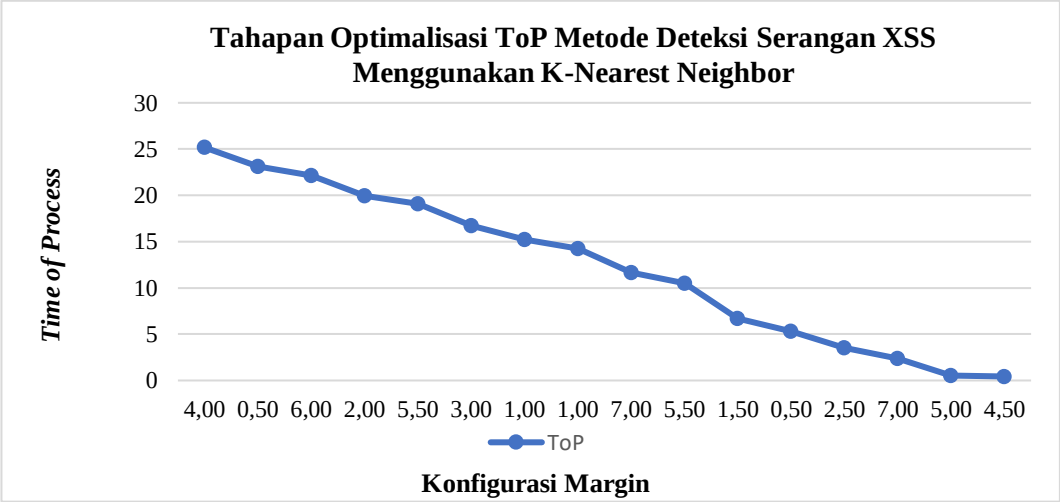
Grafik 4.19. Tahapan Optimalisasi ToP Metode Deteksi XSS *Gradient Boosting*

Senada dengan *Logistic Regression*, *Gradient Boosting* juga menunjukan pola gerakan optimalisasi yang sama. Namun, tingkat akurasi yang dihasilkan *Gradient Boosting* masih berada di bawah *Logistic Regression*, yaitu 0,9936 dan ToP 0,0001 dengan margin 5,5. Padahal ukuran margin berada di angka 5,5, lebih tinggi sebesar 0,5 dibandingkan dengan margin yang digunakan pada *Logistic Regression*, SVM, dan KN yang mencapai tingkat akurasi tertinggi dengan margin 5.

4) Tingkat Akurasi dan ToP *K-Nearest Neigbor*



Grafik 4.20. Tahapan Optimalisasi Akurasi Metode Deteksi XSS *K-Nearest Neighbor*

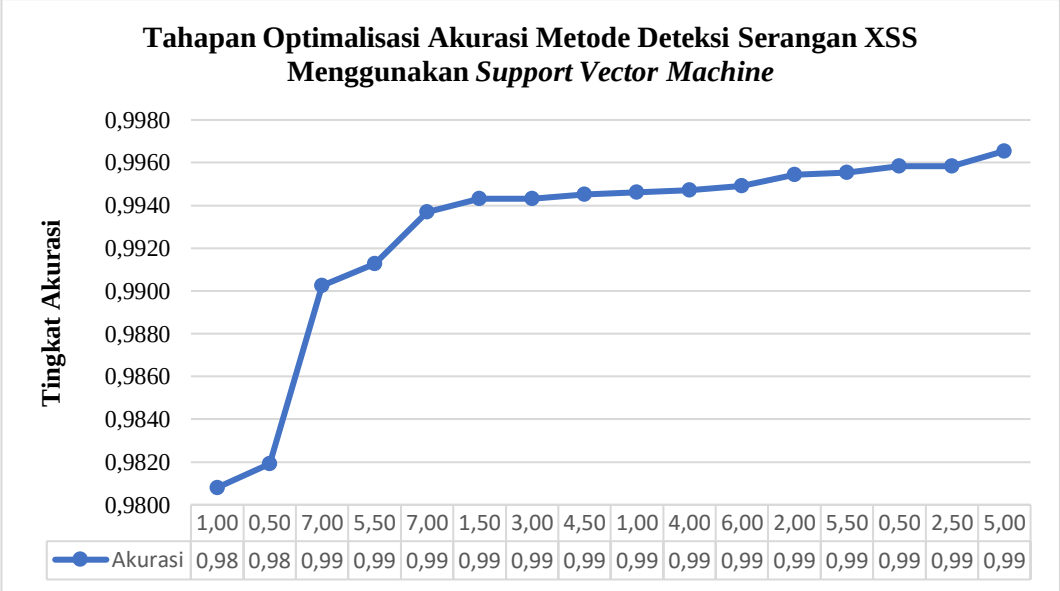


Grafik 4.21. Tahapan Optimalisasi ToP Metode Deteksi XSS *K-Nearest Neighbor*

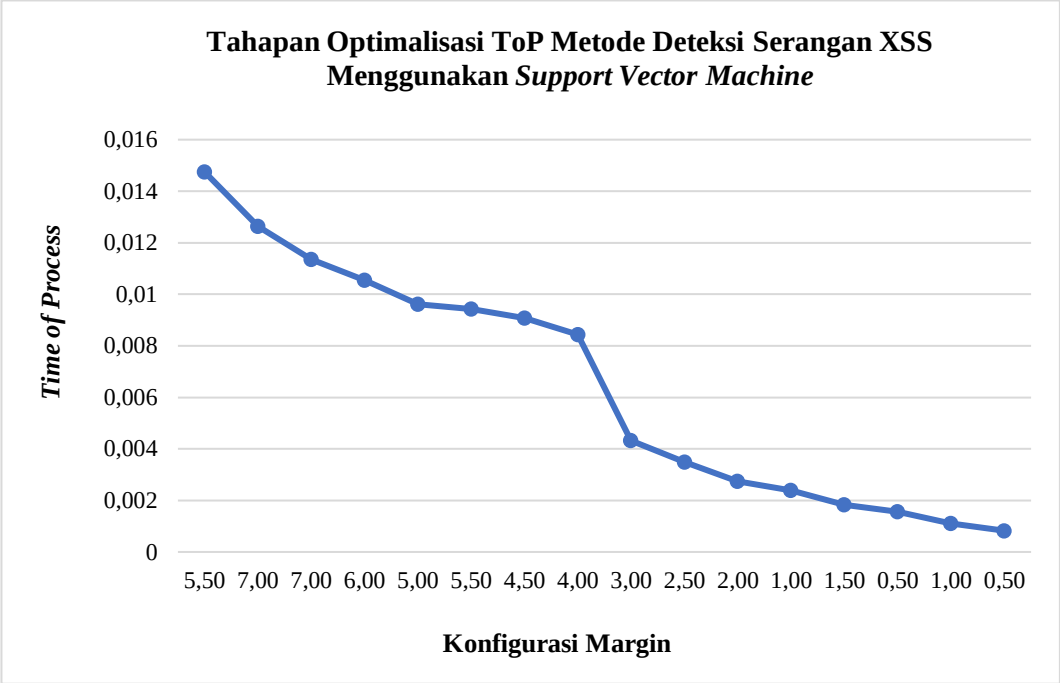
Tingkat akurasi yang didapatkan KNN sebenarnya berada pada posisi yang cukup tinggi, yaitu mencapai 0,9960. Tingkat akurasi tersebut didapatkan hanya dengan ukuran *margin* 2. Namun, dengan ukuran margin tersebut, KNN memerlukan ToP yang cukup tinggi atau berada di atas batas toleransi, yaitu 19,97 detik. Bahkan, ketika nilai margin diubah menjadi 4, ToP dapat mencapai 25.17 detik per *query*. Dengan ToP sebesar itu, KNN melakukan deteksi data lebih lambat dari lainnya, sehingga tidak efisien digunakan oleh metode deteksi.

5) Tingkat Akurasi dan ToP *Support Vector Machine*

Dari lima algoritma yang telah diujikan, SVM menjadi algoritma pilihan karena mampu mencapai tingkat akurasi sebesar 0,9965 dengan ToP 0,00096. Selain itu, tingkat akurasi terendah yang didapatkan SVM juga terbilang cukup tinggi jika dibandingkan dengan yang lainnya. Dengan ukuran *margin* 0,5, SVM mampu mendapatkan tingkat akurasi 0,9808 dengan ToP 0,0001. Setelah dilakukan tahapan optimalisasi lanjutan, tingkat akurasinya juga meningkat cukup signifikan.



Grafik 4.22. Tahapan Optimalisasi Akurasi Metode Deteksi XSS SVM



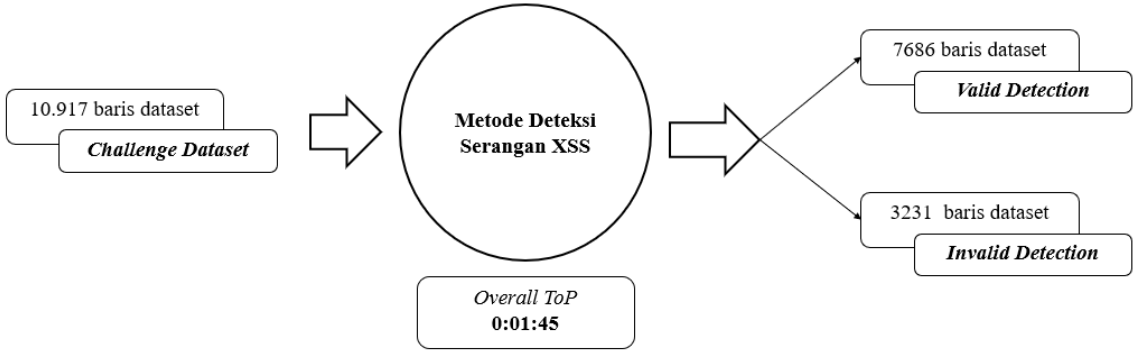
Grafik 4.23.. Tahapan Optimalisasi ToP Metode Deteksi XSS SVM

Jika dilihat dari karakteristik datasetnya, dataset serangan XSS memiliki karakter yang lebih panjang dari dataset SQLi dan RCE. Panjang karakter tersebut dapat memengaruhi kecepatan algoritma dalam melakukan deteksi. Semakin panjang karakter yang dideteksi, semakin lama atau tinggi angka ToPnya. Selain itu, metode deteksi serangan XSS sebaiknya diletakan pada beberapa lapisan, karena serangan XSS terdiri dari tiga jenis, yaitu *reflected, stored*, dan DOM.

Untuk *reflected XSS*, metode deteksi dapat diletak-an pada lapisan *HTTP Midleware*. Sementara untuk *stored* dan DOM XSS, metode deteksi dapat diletakan pada komponen *parser* atau fungsi yang dapat dipanggil sebelum laman *frontend* menampilkan data dari database. Dengan menempatkan metode deteksi pada lapisan yang relevan dengan jenis serangannya, tingkat keamanan *website* dapat semakin meningkat, terutama terkait dengan teknik serangan XSS.

E. *Performance Testing*

Untuk mendapatkan metode deteksi serangan siber yang akurat, tingkat akurasi saja tidak cukup. Kinerja atau performa metode harus diuji lebih komprehensif. Oleh karena itu, tahapan lanjutan yang dilakukan adalah pengujian kinerja metode deteksi dengan dataset serangan XSS tambahan. Melalui proses ini, performa metode deteksi dapat terukur lebih representatif. Setelah tingkat akurasi masing-masing algoritma diujikan, algoritma tersebut dikemas menjadi *Python Object* agar dapat digunakan untuk mendeteksi serangan XSS menggunakan dataset baru yang disebut *challenge dataset*.



Gambar 4.20 Implementasi Metode Deteksi SQLi pada *Challenge Dataset*

Berdasarkan *performance testing* yang telah dilakukan, metode terpilih berhasil mendeteksi 70% (ID 51 pada tabel 4.25) serangan secara valid dengan rincian 7686 *valid* dan 3231 *invalid.* Untuk menyelesaikan 30% prediksi serangan *invalid* tersebut, metode mitigasi diimplementasikan. Efektivitas metode mitigasi dijelaskan secara tersendiri pada penjelasan berikutnya.

Tabel 4.25. Pengujian Performa Metode Deteksi *Dataset Challenge* XSS

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Seconds*	*Chal. Top Per Item*	*Margin*	*Corpus Rules (L.A.R)*
1	Support Vector Machine	7727	3190	71%	0:00:09	9	0,000824402	0,5	TFF
2	K-Nearest Neigbors	7632	3285	70%	16:06:88	58048	5,317211688	0,5	TFF
3	Gradient Boosting	7798	3119	71%	0:00:06	6	0,000549602	0,5	TFF
4	Logistic Regression	7853	3064	72%	0:00:04	4	0,000366401	0,5	TFF
5	Naive Bayes	9700	1217	89%	0:00:03	3	0,000274801	0,5	TFF
6	K-Nearest Neigbors	10079	838	92%	70:07:13	252433	23,12292754	0,5	TTT
7	Support Vector Machine	9782	1135	90%	0:00:17	17	0,001557204	0,5	TTT
8	Logistic Regression	9727	1190	89%	0:00:04	4	0,000366401	0,5	TTT
9	Gradient Boosting	9767	1150	89%	0:00:07	7	0,000641202	0,5	TTT
10	Naive Bayes	8403	2514	77%	0:00:03	3	0,000274801	0,5	TTT
11	Support Vector Machine	7707	3210	71%	0:00:12	12	0,001099203	1	TFF
12	K-Nearest Neigbors	7649	3268	70%	46:12:47	166367	15,23925987	1	TFF
13	Gradient Boosting	7723	3194	71%	0:00:07	7	0,000641202	1	TFF
14	Logistic Regression	7854	3063	72%	0:00:05	5	0,000458001	1	TFF
15	Support Vector Machine	9732	1185	89%	0:00:26	26	0,002381607	1	TTT
16	Logistic Regression	9787	1130	90%	0:00:05	5	0,000458001	1	TTT
17	Gradient Boosting	8776	2141	80%	0:00:07	7	0,000641202	1	TTT

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Seconds*	*Chal. Top Per Item*	*Margin*	*Corpus Rules (L.A.R)*
18	Naive Bayes	8458	2459	77%	0:00:05	5	0,000458001	1	TFF
19	K-Nearest Neigbors	9212	1705	84%	43:13:31	155611	14,25400751	1	TTT
20	Naive Bayes	8200	2717	75%	0:00:05	5	0,000458001	1	TTT
21	K-Nearest Neigbors	7511	3406	69%	20:21:75	73335	6,717504809	1,5	TFF
22	Support Vector Machine	7647	3270	70%	0:00:20	20	0,001832005	1,5	TFF
23	Gradient Boosting	7693	3224	70%	0:00:06	6	0,000549602	1,5	TFF
24	Logistic Regression	7917	3000	73%	0:00:05	5	0,000458001	1,5	TFF
25	Naive Bayes	8331	2586	76%	0:00:06	6	0,000549602	1,5	TFF
26	K-Nearest Neigbors	7508	3409	69%	60:34:01	218041	19,97261152	2	TFF
27	Support Vector Machine	7650	3267	70%	0:00:30	30	0,002748008	2	TFF
28	Logistic Regression	7947	2970	73%	0:00:06	6	0,000549602	2	TFF
29	Gradient Boosting	7548	3369	69%	0:00:09	9	0,000824402	2	TFF
30	Naive Bayes	8206	2711	75%	0:00:08	8	0,000732802	2	TFF
31	K-Nearest Neigbors	7512	3405	69%	10:41:37	38497	3,526335074	2,5	TFF
32	Support Vector Machine	7688	3229	70%	0:00:38	38	0,00348081	2,5	TFF
33	Gradient Boosting	7628	3289	70%	0:00:07	7	0,000641202	2,5	TFF
34	Logistic Regression	7939	2978	73%	0:00:05	5	0,000458001	2,5	TFF
35	Naive Bayes	8100	2817	74%	0:00:08	8	0,000732802	2,5	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Seconds*	*Chal. Top Per Item*	*Margin*	*Corpus Rules (L.A.R)*
36	K-Nearest Neigbors	7492	3425	69%	50:42:16	182536	16,72034442	3	TFF
37	Support Vector Machine	7686	3231	70%	0:00:47	47	0,004305212	3	TFF
38	Logistic Regression	8004	2913	73%	0:00:06	6	0,000549602	3	TFF
39	Gradient Boosting	7698	3219	71%	0:00:07	7	0,000641202	3	TFF
40	Naive Bayes	8062	2855	74%	0:00:09	9	0,000824402	3	TFF
41	Support Vector Machine	7771	3146	71%	0:01:32	92	0,008427224	4	TFF
42	K-Nearest Neigbors	7537	3380	69%	76:19:47	274787	25,17055968	4	TFF
43	Logistic Regression	8170	2747	75%	0:00:09	9	0,000824402	4	TFF
44	Gradient Boosting	7734	3183	71%	0:00:11	11	0,001007603	4	TFF
45	Naive Bayes	8019	2898	73%	0:00:14	14	0,001282404	4	TFF
46	Support Vector Machine	7731	3186	71%	0:01:39	99	0,009068425	4,5	TFF
47	K-Nearest Neigbors	7542	3375	69%	1:18:75	4755	0,43555922	4,5	TFF
48	Logistic Regression	8158	2759	75%	0:00:08	8	0,000732802	4,5	TFF
49	Gradient Boosting	7705	3212	71%	0:00:09	9	0,000824402	4,5	TFF
50	Naive Bayes	8007	2910	73%	0:00:14	14	0,001282404	4,5	TFF
51	**Support Vector Machine**	**7686**	**3231**	**70%**	**0:01:45**	**105**	**0,009618027**	**5**	**TFF**
52	K-Nearest Neigbors	7547	3370	69%	1:39:09	5949	0,544929926	5	TFF
53	Logistic Regression	8024	2893	74%	0:00:09	9	0,000824402	5	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Seconds*	*Chal. Top Per Item*	*Margin*	*Corpus Rules (L.A.R)*
54	Gradient Boosting	7612	3305	70%	0:00:11	11	0,001007603	5	TFF
55	Naive Bayes	7960	2957	73%	0:00:26	26	0,002381607	5	TFF
56	Support Vector Machine	7809	3108	72%	0:01:43	103	0,009434826	5,5	TFF
57	K-Nearest Neigbors	7499	3418	69%	31:46:03	114363	10,47568013	5,5	TFF
58	Logistic Regression	7963	2954	73%	0:00:09	9	0,000824402	5,5	TFF
59	Gradient Boosting	7684	3233	70%	0:00:11	11	0,001007603	5,5	TFF
60	Logistic Regression	9145	1772	84%	0:00:09	9	0,000824402	5,5	TTT
61	Support Vector Machine	8397	2520	77%	0:02:41	161	0,014747641	5,5	TTT
62	K-Nearest Neigbors	8223	2694	75%	57:54:3	208443	19,09343226	5,5	TTT
63	Gradient Boosting	8959	1958	82%	0:00:11	11	0,001007603	5,5	TTT
64	Naive Bayes	7971	2946	73%	0:00:17	17	0,001557204	5,5	TFF
65	Naive Bayes	7892	3025	72%	0:00:23	23	0,002106806	5,5	TTT
66	Support Vector Machine	7762	3155	71%	0:01:55	115	0,010534029	6	TFF
67	Logistic Regression	7998	2919	73%	0:00:10	10	0,000916003	6	TFF
68	K-Nearest Neigbors	7467	3450	68%	67:09:27	241767	22,14591921	6	TFF
69	Gradient Boosting	7690	3227	70%	0:00:13	13	0,001190803	6	TFF
70	Naive Bayes	7975	2942	73%	0:00:29	29	0,002656407	6	TFF
71	K-Nearest Neigbors	7390	3527	68%	35:18:44	127124	11,644591	7	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Seconds*	*Chal. Top Per Item*	*Margin*	*Corpus Rules (L.A.R)*
72	Support Vector Machine	7719	3198	71%	0:02:04	124	0,011358432	7	TFF
73	Logistic Regression	7873	3044	72%	0:00:11	11	0,001007603	7	TFF
74	Gradient Boosting	7537	3380	69%	0:00:12	12	0,001099203	7	TFF
75	Support Vector Machine	8499	2418	78%	0:02:18	138	0,012640835	7	TTT
76	Logistic Regression	9167	1750	84%	0:00:10	10	0,000916003	7	TTT
77	K-Nearest Neigbors	7816	3101	72%	7:08:86	25766	2,360172208	7	TTT
78	Gradient Boosting	8962	1955	82%	0:00:11	11	0,001007603	7	TTT
79	Naive Bayes	7922	2995	73%	0:00:21	21	0,001923605	7	TFF
80	Naive Bayes	7927	2990	73%	0:00:19	19	0,001740405	7	TTT

Seperti yang terlihat pada tabel tersebut, terdapat 30% dataset XSS yang belum dapat dideteksi secara valid oleh metode deteksi terpilih. Dalam prakteknya, ketika metode deteksi tidak dapat mendeteksi sekian persen serangan XSS secara valid, terdapat potensi *website* tersebut dilumpuhkan atau mendapatkan serangan lanjutan. Namun demikian, kenyataan seperti ini telah diantisipasi oleh metode yang kedua, yaitu metode mitigasi *multi-layer security*. Dalam hal ini, metode deteksi juga tidak cukup untuk menangani serangan siber. Oleh karena itu, selain metode deteksi, metode mitigasi juga diperlukan untuk semakin meningkatkan keamanan aplikasi *website*. Penanganan *dataset challenge* yang *invalid* dijelaskan secara tersendiri pada penjelasan metode mitigasi.

4.2.1.2. Teknik Serangan *SQL Injection (SQLi)*

A. *Data Preparation*

Dataset serangan SQLi terdiri dari 30.904 baris dataset. Setelah dilakukan *data pre-processing*, total baris data yang *eligible* untuk digunakan adalah 30.609, yang arti-nya terdapat 295 baris dataset yang dieliminasi. Dengan dataset berjumlah 30.609 tersebut, proses pembagian *data training* dan *data testing* sangat representatif, sehingga tingkat akurasi yang diperoleh juga sangat tinggi. Pembahasan tingkat akurasi metode deteksi serangan SQLi dijelaskan pada sub-bab tersendiri. Dalam hal ini, terkait gambaran isi dataset sebelum melalui *data preprocessing* dapat dilihat pada tabel berikut.

Tabel 4.26. *Sample* Dataset SQLi Sebelum Melalui *Data Preprocessing*

	Sentence	***Label***	***Unnamed: 2***	***Unnamed: 3***
8038	1%' and 9660 = (select count (*) ...	1	NaN	NaN
28509	SELECT * FROM burn WHERE race BETWEEN voyage09...	0	NaN	NaN
2928	1'‖ (select 'jnsd' from dual where 2316 =...	1	NaN	NaN
4609	-4705' where 2029 = 2029 union all select 20...	1	NaN	NaN
29031	SELECT DISTINCT plan FROM afternoon	0	NaN	NaN

Seperti yang tertera pada tabel 4.26 di atas, pada tahap *data preprocessing* kolom *unnamed* 2 dan *unnamed* 3 dihilangkan karena kolom tersebut tidak dipergunakan pada proses penelitian metode deteksi serangan SQLi. Kolom *label* dalam tabel tersebut yang berisi nilai 1 adalah *payload* atau serangan dan 0 adalah *non-payload* atau bukan serangan.

B. *Data Preprocessing and Modeling*

Terdapat lima tahapan *data processing* yang dilakukan pada dataset serangan SQLi. Kelima tahapan tersebut adalah (1) menghilangkan kolom-kolom yang tidak dipergunakan dalam penelitian; (2) mengonversi kolom "label" menjadi data numerik, agar mempermudah proses operasi matematis; (3) menghapus baris data yang kosong atau NaN; (4) menghapus data yang duplikat; (5) memperbaiki tipe *encoding* yang bermasalah. Berikut ini adalah *sample* dataset dengan n = 5.

Tabel 4.27. *Sample* Dataset SQLi Setelah Melalui *Data Preprocessing*

	Sentence	***Label***
11466	vinko	0
12394	denno	0
3461	1" and 4386 = utl_inaddr.get_host_address (...	1
30801	SELECT * FROM burst WHERE voyage = 'period' O...	0
30674	SELECT continent (s) FROM ice INNER JOIN	0

Data modeling yang telah dilakukan menggunakan *margin* 6,5. Dengan vektor tersebut, jumlah vektor dibatasi menjadi 104 vektor. Dalam hal ini, 104 vektor tersebut menjadi *identifier* pada dataset serangan SQLi, baik pada label *payload* dan *non-payload*. Vektor-vektor atau *identifier* yang digunakan untuk mendeteksi serangan SQLi dapat dilihat pada tabel 4.29 berikut ini.

Tabel 4.28. Daftar 100 dari 104 vektor Serangan SQLi Label *Payload*

No	**Vektor**	**Jumlah Kemunculan**
1	)	45252
2	(	40095
3	,	20265
4	=	13701

No	Vektor	Jumlah Kemunculan
5	1	11506
6	select	9999
7	'	9440
8	||	8367
9	chr	6588
10	and	6024
11	char	5629
12	"	5148
13	as	5109
14	null	5022
15	from	4297
16	where	4069
17	--	3875
18	113	3755
19	or	3564
20	*	3356
21	union	2442
22	-	2413
23	.	2401
24	0	2224
25	+	2205
26	#	1938
27	all	1849
28	end	1651
29	else	1629
30	then	1589
31	case	1586
32	when	1586
33	count	1536
34	122	1502

No	Vektor	Jumlah Kemunculan
35	5	1442
36	sysusers	1309
37	all_users	1134
38	elt	1078
39	$	1058
40	like	1032
41	rdb	1001
42	dual	895
43	t1	894
44	t2	894
45	t3	894
46	concat	875
47	sleep	872
48	112	865
49	106	807
50	0x7171706a71	783
51	0x717a767a71	783
52	118	753
53	%'	705
54	%"	703
55	sysibm	669
56	systables	669
57	domain	669
58	2	625
59	500000000	598
60	x	591
61	regexp_substring	559
62	repeat	559
63	5000000	543
64	'||	516

No	Vektor	Jumlah Kemunculan
65	65	499
66	by	463
67	t4	448
68	users	447
69	id	447
70	'+	406
71	/	402
72	"%"	396
73	'%'	392
74	69	371
75	upper	331
76	benchmark	328
77	md5	328
78	83	316
79	rlike	293
80	',	288
81	:	282
82	right	280
83	left	279
84	crypt_key	279
85	generate_series	278
86	group	277
87	version	275
88	floor	275
89	rand	275
90	a	273
91	||'	261
92	extractvalue	260
93	0x5c	260
94	in	260

No	Vektor	Jumlah Kemunculan
95	pg_sleep	238
96	information_schema	231
97	0x78	226
98	8,44674E+18	226
99	dbms_pipe	226
100	receive_message	226

Metode deteksi serangan SQLi didesain agar tidak hanya mampu mendeteksi serangan SQLi dengan label *payload* secara valid, tetapi juga label *non-payload*. Dalam kaitannya dengan *confussion matrix*, nilai dari TP dan TN harus sama-sama tinggi. Dengan demikian, metode deteksi serangan SQLi dapat diimplementasikan pada arsitektur *multi-layer security* secara optimal. Oleh karena itu, vektor-vektor yang digunakan pada label *non-payload* juga harus representatif dengan serangan SQLi pada kehidupan nyata. Berikut ini adalah daftar 100 dari 104 vektor serangan SQLi dengan label *non-payload* yang digunakan pada meotde deteksi.

Tabel 4.29. Daftar 100 dari 104 vektor Serangan SQLi Label *Non-Payload*

No	Vektor	Jumlah Kemunculan
1	select	12496
2	from	11860
3	'	10468
4	*	6356
5	(	5540
6	)	5430
7	,	4893
8	where	4308
9	=	4034
10	.	3436
11	3	1543

No	Vektor	Jumlah Kemunculan
12	',	1540
13	-	1468
14	and	1354
15	as	1304
16	s	1288
17	by	1213
18	not	1104
19	order	1087
20	join	1034
21	top	979
22	@	911
23	count	894
24	between	870
25	first	859
26	"	845
27	fetch	837
28	rows	837
29	orders	813
30	+	795
31	/	773
32	percent	747
33	avg	745
34	50	735
35	employees	663
36	only	636
37	union	563
38	in	557
39	on	496
40	like	452
41	sum	449
42	7	430
43	1996	428
44	wp_posts	392
45	limit	390

No	Vektor	Jumlah Kemunculan
46	or	355
47	id	332
48	1	332
49	;	327
50	31	326
51	distinct	325
52	asc	323
53	1	322
54	de	311
55	city	287
56	10	286
57	post_id	271
58	c	267
59	is	255
60	left	248
61	inner	243
62	all	238
63	into	237
64	null	235
65	desc	227
66	insert	226
67	20	224
68	price	223
69	values	222
70	update	220
71	set	219
72	%'	217
73	delete	216
74	min	215
75	onlyselect	214
76	'[	214
77	]%'	214
78	calle	213
79	?	205

No	Vektor	Jumlah Kemunculan
80	wp_postmeta	153
81	:	143
82	meta_key	142
83	meta_value	140
84	[	135
85	meta_id	134
86	column_name	129
87	right	129
88	full	128
89	outer	128
90	option_name	125
91	wp_options	124
92	option_value	122
93	employeeid	119
94	customers	117
95	]	115
96	ordersinner	111
97	",	108
98	ordersright	108
99	'.	108
100	3select	107

Sama halnya dengan vektor-vektor yang digunakan pada serangan XSS, beberapa vektor pada serangan SQLi juga saling beririsan. Fenomena irisan ini terjadi ketika suatu vektor berada pada kedua label, yaitu *payload* dan juga *non-payload*. Berikut ini adalah vektor-vektor yang saling beririsan pada dataset serangan SQLi.

Tabel 4.30. Irisan Vektor Label Payload dan Non-Payload Dataset SQLi

No	*Vektor*	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
1	)	45252	5430	39822	payload
2	(	40095	5540	34555	payload
3	,	20265	4893	15372	payload

No	*Vektor*	*Payload*	*Non-Payload*	*Degree to Payload*	*Tendency*
4	=	13701	4034	9667	payload
5	1	11506	322	11184	payload
6	select	9999	12496	-2497	non-payload
7	'	9440	10468	-1028	non-payload
8	and	6024	1354	4670	payload
9	"	5148	845	4303	payload
10	as	5109	1304	3805	payload
11	null	5022	235	4787	payload
12	from	4297	11860	-7563	non-payload
13	where	4069	4308	-239	non-payload
14	or	3564	355	3209	payload
15	*	3356	6356	-3000	non-payload
16	union	2442	563	1879	payload
17	-	2413	1468	945	payload
18	.	2401	3436	-1035	non-payload
19	+	2205	795	1410	payload
20	all	1849	238	1611	payload
21	count	1536	894	642	payload
22	like	1032	452	580	payload
23	%'	705	217	488	payload
24	by	463	1213	-750	non-payload
25	id	447	332	115	payload
26	/	402	773	-371	non-payload
27	',	288	1540	-1252	non-payload
28	:	282	143	139	payload
29	right	280	129	151	payload
30	left	279	248	31	payload
31	in	260	557	-297	non-payload

Terdapat 31 vektor yang beririsan pada dataset serangan SQLi. Irisan vektor dataset seperti tabel 4.31 tersebut memungkinkan terjadinya prediksi *False Positive* dan *False Negative*, yaitu ketika metode deteksi keliru atau salah dalam melakukan prediksi. Oleh karena itu, kalkulasi tendensi dilakukan sehingga kecenderungan vektor yang beririsan tersebut dapat diukur posisi atau klasifikasinya, apakah lebih cenderung pada label *payload* atau *non-payload*.

C. *Feature Enginering, Data Training, and Data Testing*

Seperti yang telah disinggung pada penjelasan sebelumnya, serangan XSS, SQLi, dan RCE adalah serangan berbasis *string* atau teks, sehingga *feature engineering* yang diterapkan pada penelitian ini melibatkan fungsi *text processing*. Dalam hal ini, serangan SQLi juga menerapkan dua tahapan *feature enginering*, yaitu (1) kontruksi corpus dan (2) kontruksi fitur. Alur kontruksi corpus dapat dilihat pada gambar 4.27 dan 4.28, dan kontruksi fitur dapat dilihat pada gambar 4.29.

Sama halnya dengan distribusi persentase *data training* dan *data testing* pada serangan XSS, *data testing* serangan SQLi juga menggunakan nilai 0,2 sebagai *test size*, yang artinya 20% dataset digunakan sebagai *data testing*. Penentuan *test size* atau persentase data uji tersebut bukan tanpa dasar. Dalam tahapan optimalisasi kinerja metode deteksi, ukuran *test size* 0,1 juga diimplementasikan, tidak hanya 0,2 *test size*. Hal ini dilakukan untuk mendapatkan ukuran *test size* yang paling optimal. Berdasarkan tahapan-tahapan eksperimen yang telah dilakukan, ukuran *test size* sebesar 0,2 tersebut terbukti dapat lebih representatif. Artinya, dengan dataset sebesar 20% tersebut, metode deteksi dapat memiliki data uji yang lebih komprehensif untuk menguji atau mengukur tingkat akurasi deteksinya dibandingkan hanya 10%.

D. *Performance Optimalization*

Sesuai dengan yang telah direncanakan, penelitian ini mengajukan metode deteksi serangan siber menggunakan *machine learning* tanpa menyebutkan secara spesifik algoritma yang digunakan. Hal tersebut dilakukan karena tiap serangan siber memiliki karakteristik yang berbeda-beda, sehingga penentuan algoritma di awal, apalagi tanpa pengujian, dapat mengurangi kualitas akurasi metode deteksi. Selain itu, penelitian ini berupaya selektif dalam mencari algoritma yang benar-benar tepat digunakan dalam metode deteksi. Oleh karena itu, terdapat lima algoritma yang diujikan, dan kelimanya dipilih berdasarkan data-data dari penelitian sebelumnya. Kelima algoritma tersebut dioptimalisasikan kinerjanya, sehingga peneliti dapat menemukan formulasi terbaik dalam metode deteksi serangan SQLi. Berikut ini adalah tabel data rekam jejak upaya optimalisasi kinerja masing-masing algoritma yang mendasari pemilihan **SVM** sebagai algoritma yang digunakan untuk melakukan deteksi serangan SQLi. Tabel diurutkan berdasarkan tingkat akurasi.

Tabel 4.31. Tahapan Optimalisasi Kinerja Metode Deteksi SQLi Lima Algoritma

ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
1	**Support Vector Machine**	**0,99771**	**0:00:10**	**6.5**	**104**	**6122**	**24487**	**True**	**False**	**False**
2	K-Nearest Neigbors	0,99706	0:01:23	6.5	104	6122	24487	True	False	False
3	Support Vector Machine	0,99722	0:00:13	10	160	6122	24487	True	False	False
4	K-Nearest Neigbors	0,99673	0:01:06	10	160	6122	24487	True	False	False

ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
5	Support Vector Machine	0,99641	0:00:02	1.5	24	6122	24487	True	False	False
6	Support Vector Machine	0,99641	0:00:15	7.5	120	6122	24487	True	False	False
7	K-Nearest Neigbors	0,99624	0:00:12	1.5	24	6122	24487	True	False	False
8	Support Vector Machine	0,99624	0:00:12	9.5	152	6122	24487	True	False	False
9	Support Vector Machine	0,99624	0:00:08	5.5	88	6122	24487	True	False	False
10	K-Nearest Neigbors	0,99608	0:01:10	8	128	6122	24487	True	False	False
11	K-Nearest Neigbors	0,99608	0:00:42	4	64	6122	24487	True	False	False
12	Support Vector Machine	0,99592	0:00:14	8	128	6122	24487	True	False	False
13	K-Nearest Neigbors	0,99592	0:01:03	9.5	152	6122	24487	True	False	False
14	K-Nearest Neigbors	0,99592	0:00:47	5.5	88	6122	24487	True	False	False
15	Support Vector Machine	0,99592	0:00:08	6	96	6122	24487	True	False	False
16	Support Vector Machine	0,99592	0:00:06	4.5	72	6122	24487	True	False	False
17	Support Vector Machine	0,99575	0:00:03	2.5	40	6122	24487	True	False	False
18	Support Vector Machine	0,99575	0:00:06	5	80	6122	24487	True	False	False
19	K-Nearest Neigbors	0,99575	0:00:52	5	80	6122	24487	True	False	False
20	K-Nearest Neigbors	0,99559	0:01:20	7.5	120	6122	24487	True	False	False
21	K-Nearest Neigbors	0,99559	0:01:01	6	96	6122	24487	True	False	False
22	Logistic Regression	0,99543	0:00:05	9.5	152	6122	24487	True	False	False
23	Support Vector Machine	0,99543	0:00:05	4	64	6122	24487	True	False	False
24	Logistic Regression	0,99526	0:00:06	10	160	6122	24487	True	False	False

ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
25	K-Nearest Neigbors	0,99526	0:00:20	2.5	40	6122	24487	True	False	False
26	Support Vector Machine	0,99526	0:00:12	7	112	6122	24487	True	False	False
27	Logistic Regression	0,99510	0:00:04	7.5	120	6122	24487	True	False	False
28	Support Vector Machine	0,99510	0:00:15	9	144	6122	24487	True	False	False
29	K-Nearest Neigbors	0,99510	0:00:23	2	32	6122	24487	True	False	False
30	K-Nearest Neigbors	0,99510	0:00:38	4.5	72	6122	24487	True	False	False
31	Logistic Regression	0,99608	0:00:04	6.5	104	6122	24487	True	False	False
32	Gradient Boosting	0,99477	0:00:17	9.5	152	6122	24487	True	False	False
33	Gradient Boosting	0,99477	0:00:18	10	160	6122	24487	True	False	False
34	K-Nearest Neigbors	0,99461	0:01:10	7	112	6122	24487	True	False	False
35	K-Nearest Neigbors	0,99461	0:01:21	9	144	6122	24487	True	False	False
36	Support Vector Machine	0,99461	0:00:02	2	32	6122	24487	True	False	False
37	Logistic Regression	0,99445	0:00:05	8	128	6122	24487	True	False	False
38	Gradient Boosting	0,99428	0:00:15	7.5	120	6122	24487	True	False	False
39	Gradient Boosting	0,99428	0:00:10	6	96	6122	24487	True	False	False
40	K-Nearest Neigbors	0,99428	0:00:31	3.5	56	6122	24487	True	False	False
41	Gradient Boosting	0,99412	0:00:05	2.5	40	6122	24487	True	False	False
42	K-Nearest Neigbors	0,99412	0:00:34	3	48	6122	24487	True	False	False
43	Logistic Regression	0,99412	0:00:03	6	96	6122	24487	True	False	False
44	Gradient Boosting	0,99477	0:00:13	6.5	104	6122	24487	True	False	False

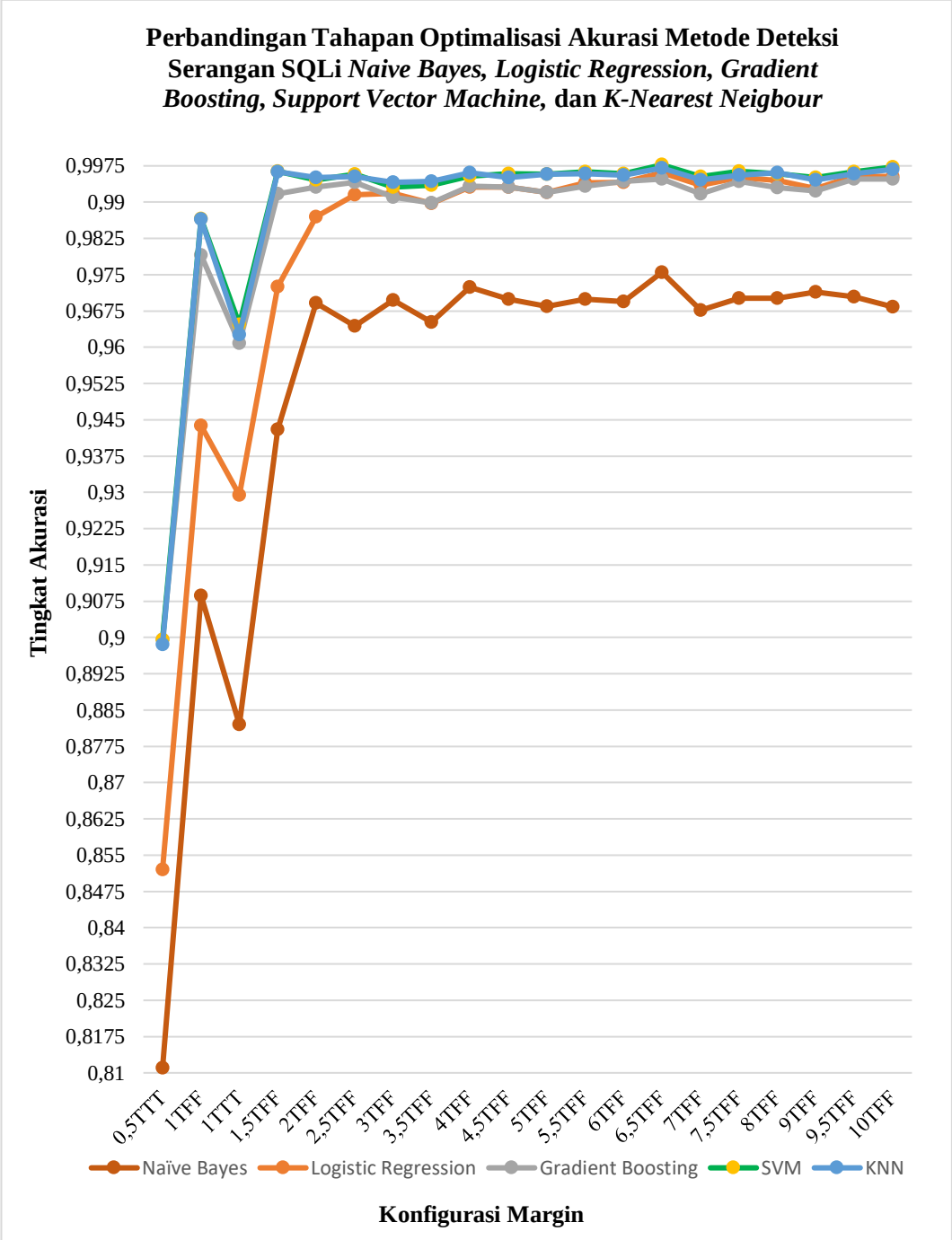
ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
45	Logistic Regression	0,99396	0:00:03	5.5	88	6122	24487	True	False	False
46	Logistic Regression	0,99347	0:00:05	7	112	6122	24487	True	False	False
47	Support Vector Machine	0,99347	0:00:05	3.5	56	6122	24487	True	False	False
48	Gradient Boosting	0,99330	0:00:10	5.5	88	6122	24487	True	False	False
49	Gradient Boosting	0,99330	0:00:07	4	64	6122	24487	True	False	False
50	Support Vector Machine	0,99314	0:00:03	3	48	6122	24487	True	False	False
51	Gradient Boosting	0,99314	0:00:04	2	32	6122	24487	True	False	False
52	Logistic Regression	0,99314	0:00:02	4.5	72	6122	24487	True	False	False
53	Gradient Boosting	0,99314	0:00:08	4.5	72	6122	24487	True	False	False
54	Logistic Regression	0,99314	0:00:02	4	64	6122	24487	True	False	False
55	Gradient Boosting	0,99298	0:00:15	8	128	6122	24487	True	False	False
56	Logistic Regression	0,99281	0:00:05	9	144	6122	24487	True	False	False
57	Gradient Boosting	0,99232	0:00:18	9	144	6122	24487	True	False	False
58	Logistic Regression	0,99200	0:00:02	5	80	6122	24487	True	False	False
59	Gradient Boosting	0,99200	0:00:09	5	80	6122	24487	True	False	False
60	Logistic Regression	0,99183	0:00:01	3	48	6122	24487	True	False	False
61	Gradient Boosting	0,99167	0:00:03	1.5	24	6122	24487	True	False	False
62	Gradient Boosting	0,99167	0:00:13	7	112	6122	24487	True	False	False
63	Logistic Regression	0,99151	0:00:01	2.5	40	6122	24487	True	False	False
64	Gradient Boosting	0,99102	0:00:05	3	48	6122	24487	True	False	False

ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
65	Gradient Boosting	0,98987	0:00:06	3.5	56	6122	24487	True	False	False
66	Logistic Regression	0,98971	0:00:01	3.5	56	6122	24487	True	False	False
67	Logistic Regression	0,98693	0:00:01	2	32	6122	24487	True	False	False
68	Support Vector Machine	0,98661	0:00:03	1	16	6122	24487	True	False	False
69	K-Nearest Neigbors	0,98644	0:00:15	1	16	6122	24487	True	False	False
70	Gradient Boosting	0,97909	0:00:02	1	16	6122	24487	True	False	False
71	Naive Bayes	0,97550	0:00:04	6.5	104	6122	24487	True	False	False
72	Logistic Regression	0,97256	0:00:00	1.5	24	6122	24487	True	False	False
73	Naive Bayes	0,97239	0:00:02	4	64	6122	24487	True	False	False
74	Naive Bayes	0,97141	0:00:05	9	144	6122	24487	True	False	False
75	Naive Bayes	0,97043	0:00:05	9.5	152	6122	24487	True	False	False
76	Naive Bayes	0,97011	0:00:04	7.5	120	6122	24487	True	False	False
77	Naive Bayes	0,97011	0:00:04	8	128	6122	24487	True	False	False
78	Naive Bayes	0,96994	0:00:03	5.5	88	6122	24487	True	False	False
79	Naive Bayes	0,96994	0:00:02	4.5	72	6122	24487	True	False	False
80	Naive Bayes	0,96978	0:00:01	3	48	6122	24487	True	False	False
81	Naive Bayes	0,96945	0:00:03	6	96	6122	24487	True	False	False
82	Naive Bayes	0,96913	0:00:01	2	32	6122	24487	True	False	False
83	Naive Bayes	0,96847	0:00:02	5	80	6122	24487	True	False	False
84	Naive Bayes	0,96831	0:00:06	10	160	6122	24487	True	False	False

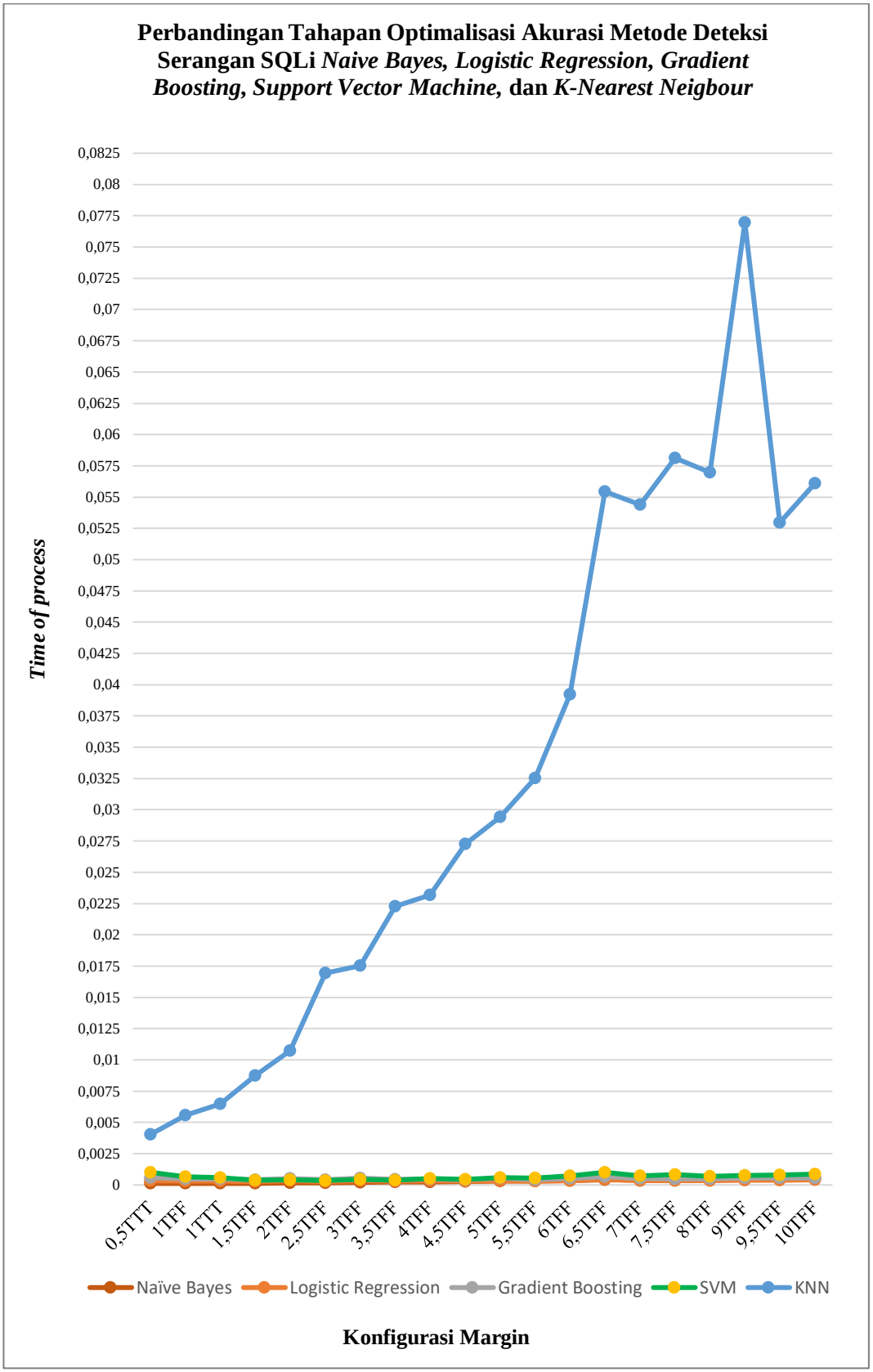
ID	*Classifier Name*	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alpha numeric*	*Remove Puncts*
85	Naive Bayes	0,96766	0:00:04	7	112	6122	24487	True	False	False
86	Naive Bayes	0,96521	0:00:01	3.5	56	6122	24487	True	False	False
87	Support Vector Machine	0,96472	0:00:07	1	16	6122	24487	True	True	True
88	Naive Bayes	0,96439	0:00:01	2.5	40	6122	24487	True	False	False
89	K-Nearest Neigbors	0,96259	0:00:19	1	16	6122	24487	True	True	True
90	Gradient Boosting	0,96080	0:00:02	1	16	6122	24487	True	True	True
91	Logistic Regression	0,94381	0:00:00	1	16	6122	24487	True	False	False
92	Naive Bayes	0,94299	0:00:00	1.5	24	6122	24487	True	False	False
93	Logistic Regression	0,92943	0:00:00	1	16	6122	24487	True	True	True
94	Naive Bayes	0,90869	0:00:00	1	16	6122	24487	True	False	False
95	Gradient Boosting	0,89954	0:00:02	0.5	8	6122	24487	True	True	True
96	Support Vector Machine	0,89954	0:00:09	0.5	8	6122	24487	True	True	True
97	K-Nearest Neigbors	0,89856	0:00:08	0.5	8	6122	24487	True	True	True
98	Naive Bayes	0,88206	0:00:00	1	16	6122	24487	True	True	True
99	Logistic Regression	0,85201	0:00:00	0.5	8	6122	24487	True	True	True
100	Naive Bayes	0,81101	0:00:00	0.5	8	6122	24487	True	True	True

Sesuai dengan tabel tersebut, SVM menjadi pilihan algoritma metode deteksi serangan SQLi karena mampu mendapatkan tingkat akurasi 0,99771. Dengan tingkat akurasi tersebut, metode deteksi sudah memiliki modal yang cukup untuk diujikan kinerja deteksinya.

Meskipun dengan dataset dan distribusi data uji yang sama, tiap algoritma deteksi memiliki karakteristik masing-masing dalam mendapatkan tingkat akurasi. Berikut ini adalah perbandingan tahapan atau upaya optimalisasi akurasi metode deteksi serangan SQLi menggunakan lima algoritma.



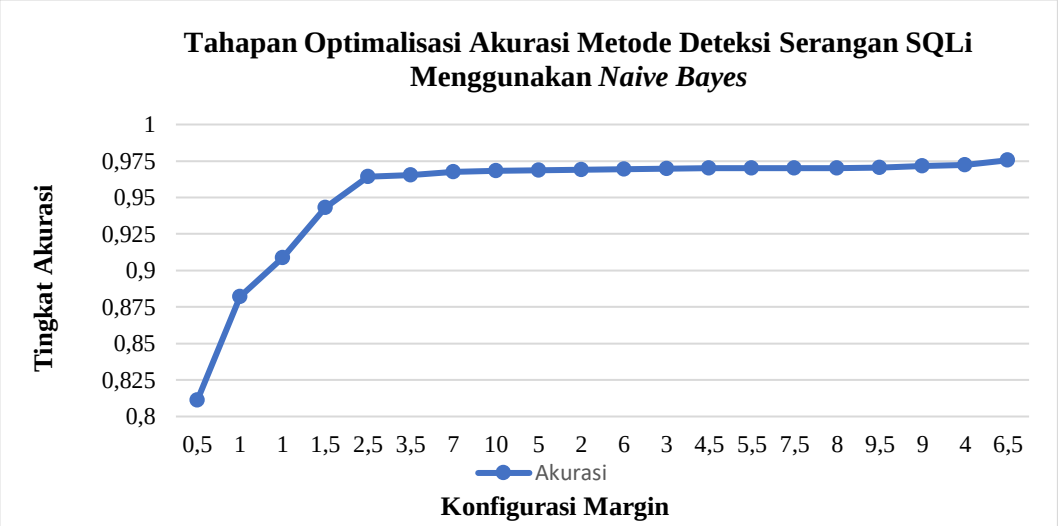
Grafik 4.24. Perbandingan Tahapan Optimalisasi Akurasi Metode Deteksi SQLi



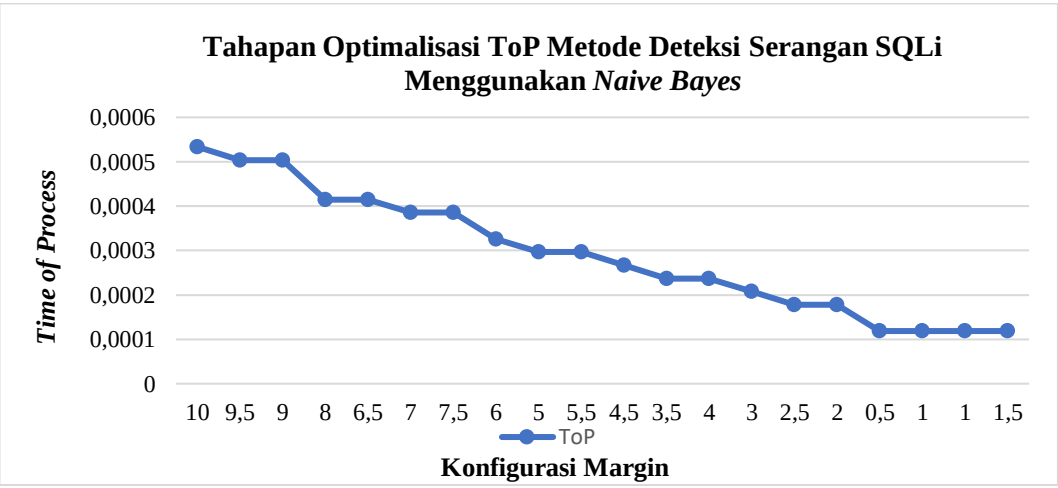
Grafik 4.25. Perbandingan Tahapan Optimalisasi ToP Metode Deteksi SQLi

Sama halnya dengan optimalisasi ToP serangan XSS, KKN memerlukan durasi waktu atau TOP yang lebih lama dibandingkan algoritma lainnya. Terlihat pada gambar 4.43, perbedaan ToP KNN dengan algoritma lainnya cukup signifikan. Meskipun tingkat akurasi yang dihasilkan KNN cukup tinggi, metode deteksi berbasis KNN terkendala dengan ToP. Dengan ToP yang tinggi, eksekusi metode deteksi berjalan lebih lambat, sehingga dikhawatirkan juga akan memperlambat kinerja *web server* dan aplikasi web yang mengimplementasikan metode ini.

1) Tingkat Akurasi dan ToP Naïve Bayes



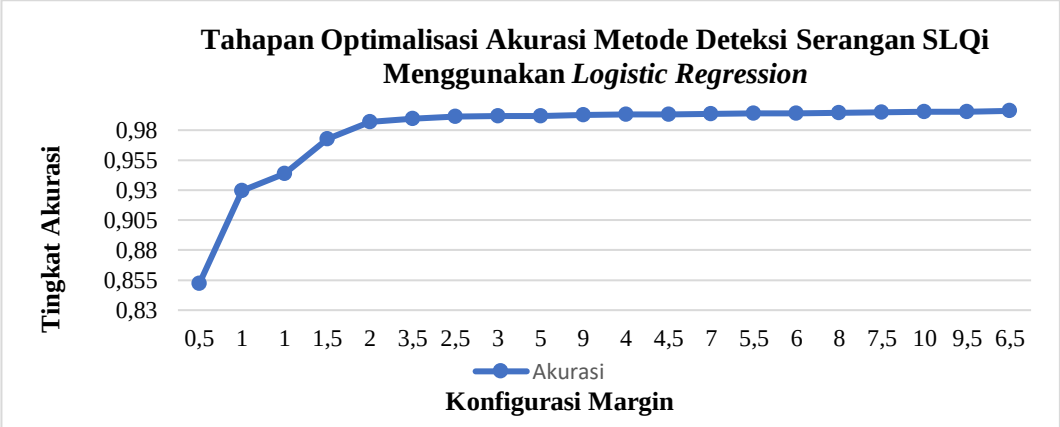
Grafik 4.26. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi Naive Bayes



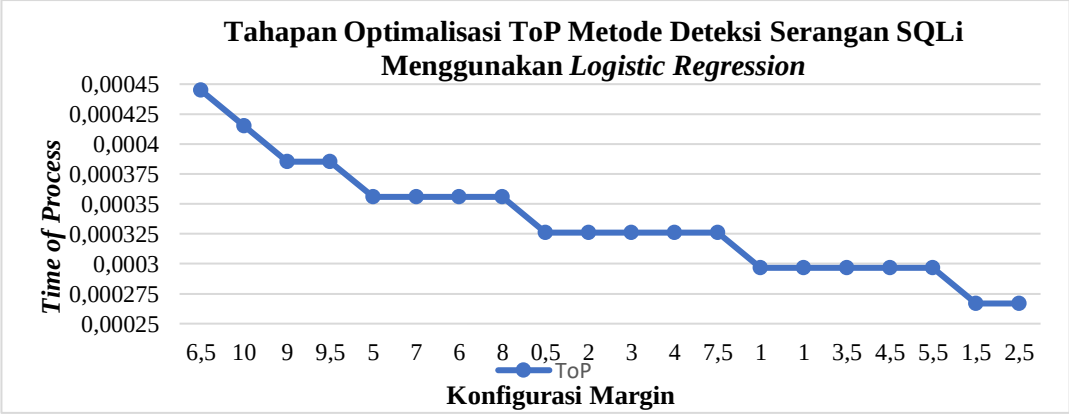
Grafik 4.27. Tahapan Optimalisasi ToP Metode Deteksi SQLi Naive Bayes

Sesuai dengan grafik 4.26 dan 4.27, tingkat akurasi tertinggi yang mampu dicapai Naïve Bayes dalam mendeteksi serangan SQLi mencapai angka 0,9754. Sementara itu, tingkat akurasi terendahnya adalah 0,8110. Tidak seperti deteksi serangan XSS, pola akurasi yang mampu dicapai Naïve Bayes dalam deteksi serangan SQLi masih lebih harmonis dengan pengaturan margin atau batas vektor yang ditentukan. Selain itu, dari sudut pandang ToP, Naïve Bayes juga terbilang aman digunakan karena ToP terlama berada pada angka, 0,00053 dengan margin 10. Dengan tingkat ToP tersebut, Naïve Bayes dapat diklasifikasikan sebagai algortima yang mampu melakukan deteksi serangan SQLi secara cepat.

2) Tingkat Akurasi dan ToP *Logistic Regression*



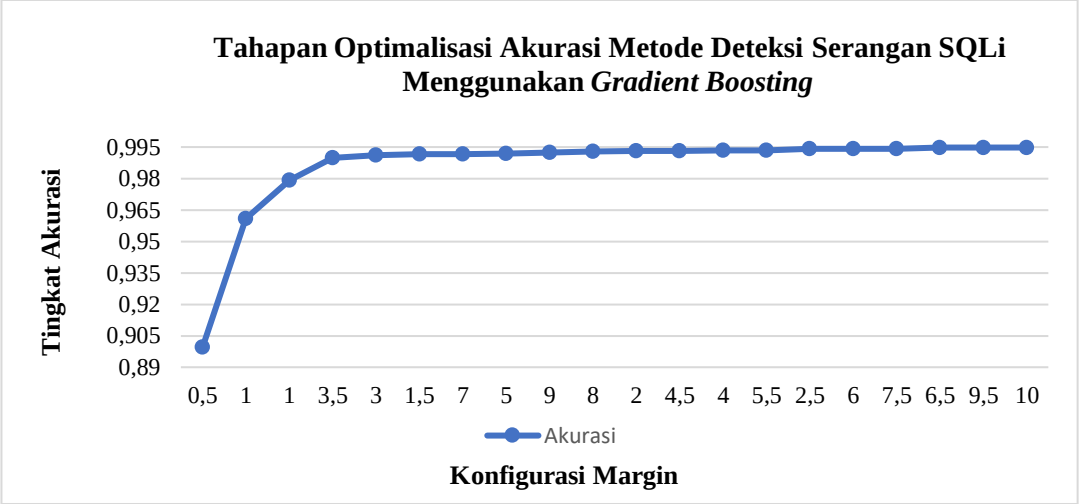
Grafik 4.28. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi *Logistic Regression*



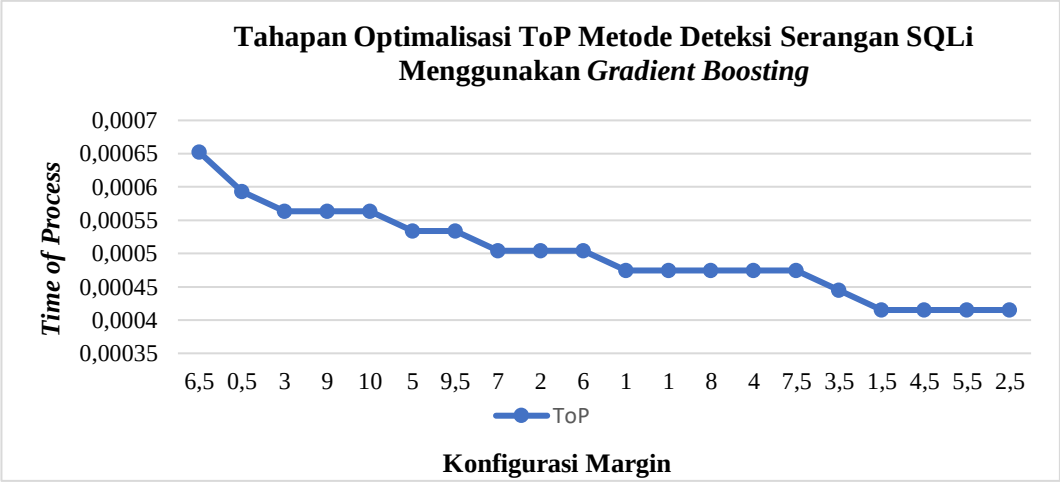
Grafik 4.29. Tahapan Optimalisasi ToP Metode Deteksi SQLi *Logistic Regression*

Tingkat akurasi tertinggi yang mampu dicapai oleh *Logistic Regression* adalah 0,9960. Tingkat akurasi tersebut diperoleh dengan pengaturan margin 6,5. Tingkat akurasi terendah algoritma ini berada pada angka 0,8520 dengan margin 0,5. Terkait dengan ToPnya, proses algoritma ini terbilang cepat. Dengan tingkat akurasi sebesar 0,9960, *Logistic Regression* memerlukan waktu ToP 0,00044. Dengan kata lain, algoritma ini terbilang memiliki tingkat akurasi dan ToP yang baik.

3) Tingkat Akurasi dan ToP *Gradient Boosting*



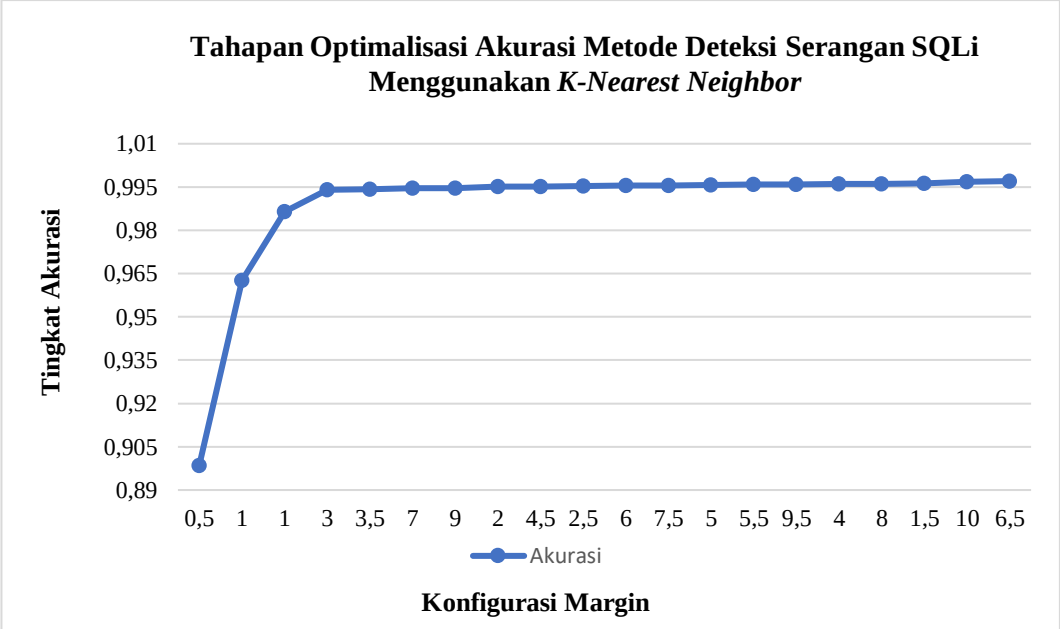
Grafik 4.30. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi *Gradient Boosting*



Grafik 4.31. Tahapan Optimalisasi ToP Metode Deteksi SQLi *Gradient Boosting*

Dalam proses pembelajaran, *Gradient Boosting* mampu mendapatkan tingkat akurasi tertinggi pada angka 0, 9947 dengan pengaturan *margin* 10. Sementara itu, tingkat akurasi terendah algoritma ini adalah 0,8995 dengan pengaturan *margin* 0,5. Dengan kata lain, algoritma ini menggunakan *margin* tertinggi (10) untuk mendapatkan tingkat akurasi tertinggi. Pada algoritma yang lain, penggunaan margin yang tinggi dapat mempengaruhi ToP. Sementara itu, pada kasus *Gradient Boosting* pada penelitian ini, hal tersebut tidak terjadi, karena ToP margin 10 *Gradient Boosting* berada di angka 0,0005, yang masih lebih rendah dibandingkan dengan margin 6,5 yang justru berada di angka 0,00065. Dengan demikian, jumlah margin atau batas vektor tidak memberi pengaruh pada jumlah atau durasi ToP.

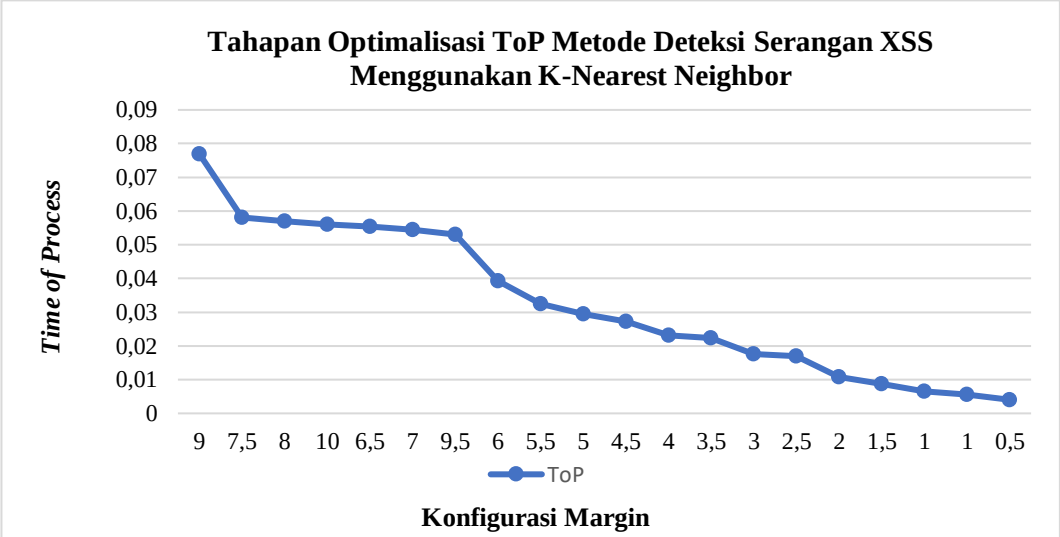
4) Tingkat Akurasi dan ToP *K-Nearest Neighbor*



Grafik 4.32. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi KNN

Tingkat akurasi tertinggi K-Nearest Neigbor mencapai angka 0,9970. Tingkat akurasi ini terbilang tinggi dan sebenarnya cukup ideal untuk diimplementasikan

pada metode deteksi. Namun demikian, algoritma ini memerlukan ToP yang cukup tinggi. Dengan jumlah *margin* terendah atau 0,5, algoritma ini menghabiskan ToP sebesar 0,0040. Bahkan, ketika margin di atur pada posisi 6,5, ToP K Nearest Neigbor semakin meningkat, berada di angka 0,0554. Sebagai gambaran, jika aplikasi web menerima 1000 *request* per detik, algoritma ini memerlukan waktu selama 55 detik untuk melakukan deteksi, sebelum aplikasi web menampilkan laman *frontend*.



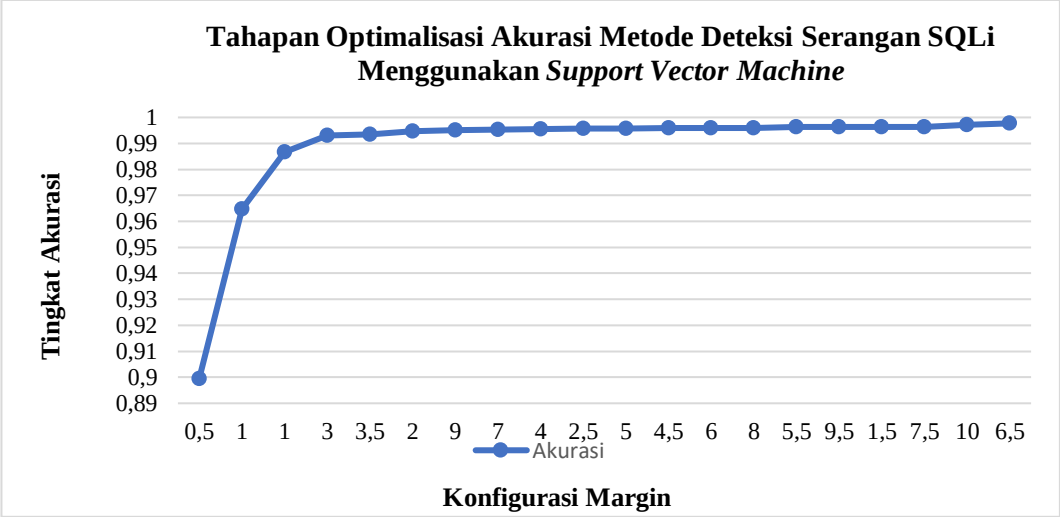
Grafik 4.33. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi KNN

Tahapan optimalisasi ToP K-Nearest Neighbor dapat dilihat pada grafik di atas. Pada grafik tersebut, durasi ToP terendah berada di angka, 0,004 dengan pengaturan *margin* 0,5. Saat *margin* berada pada posisi 9, tingkat ToP mencapai 0,07. Pola rekam jejak ToP juga terjadi pada serangan XSS, Dalam penelitian ini, KNN memerlukan durasi ToP lebih lama dibandingkan algoritma lainnya.

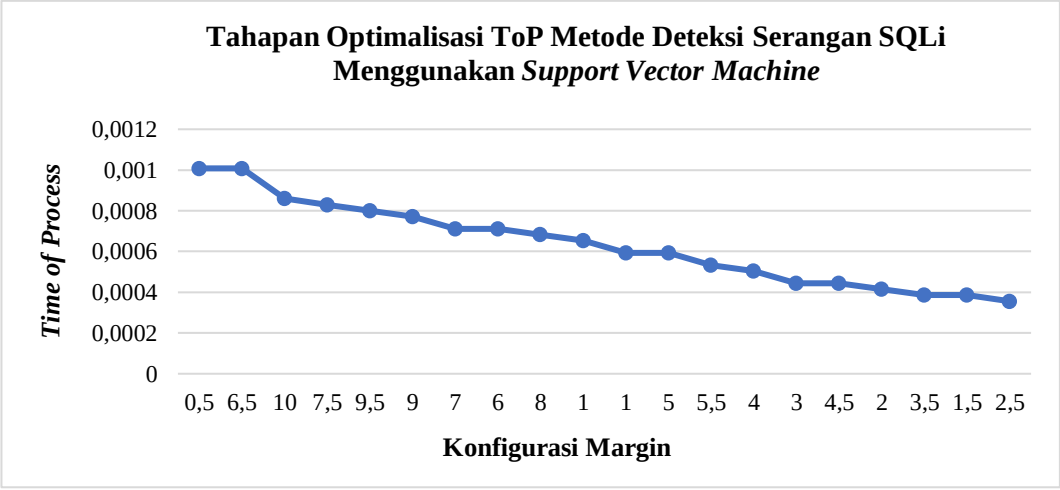
5) Tingkat Akurasi dan ToP *Support Vector Machine*

SVM menjadi algoritma yang dipilih dalam metode deteksi serangan SQLi. Pemilihan algoritma ini tentunya karena dasar yang jelas dan tegas. Berdasarkan ta-

hapan penelitian yang telah dilakukan, SVM terbukti handal dan mampu mencapai tingkat akurasi yang sangat tinggi. Bahkan, tingkat akurasi terendah SVM pada saat tahapan optimalisasi telah berada di angka 0,89. Setelah dilakukan beberapa tahap optimalisasi, bersama-sama dengan algoritma yang lain, SVM mampu mencapai tingkat akurasi tertinggi, yaitu sebesar 0,9977 dengan *margin* 6,5. Dengan margin 6,5, ToP yang diperlukan adalah 0,001. Tahap optimalisasi akurasi dan ToP SVM dapat dilihat pada gambar berikut ini.



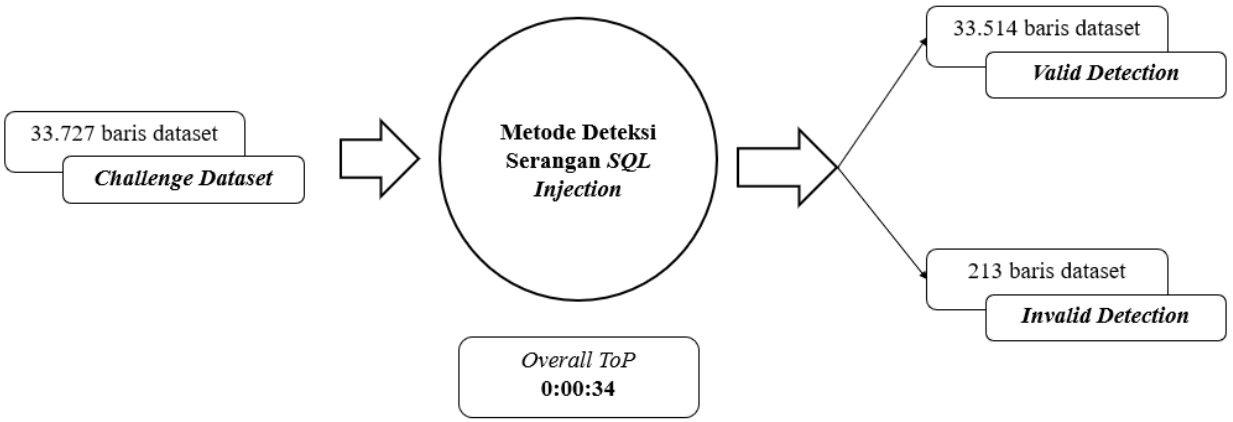
Grafik 4.34. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi SVM



Grafik 4.35.. Tahapan Optimalisasi Akurasi Metode Deteksi SQLi SVM

E. Performance *Testing*

Mengacu pada realita serangan siber saat ini, pola dan bentuk serangan SQLi dapat sangat bervariasi. Metode deteksi serangan SQLi berbasis *machine learning* harus terus diujikan kinerjanya, agar menghasilkan metode deteksi yang stabil dan konsisten dengan tingkat akurasinya. Oleh karena itu, selain melakukan penguatan pada proses belajar melalui tahapan *comprehensive data training and testing*, serta *configuration parameter switcher*, penelitian ini juga menerapkan tahapan *performance optimalization* yang disebut sebagai *challenges*. Pada tahapan ini, metode deteksi yang telah dihasilkan diuji kinerjanya dalam mendeteksi dataset yang berbeda.



Gambar 4.21. Implementasi Metode Deteksi SQLi pada *Challenge Dataset*

Jumlah *challenges dataset* yang diujikan pada metode deteksi serangan SQLi terpilih berjumlah 33.727 baris dataset. Tahapan *data preprocessing* juga dilakukan pada *challenges dataset.* Selain mengujikan pada metode deteksi terpilih, tahapan ini juga menguji-

kan algoritma lain untuk mendapatkan perbandingan kinerja dan memeroleh data ToP masing-masing algoritma. Algoritma SVM terpilih berada pada ID 66 dengan tingkat akurasi mencapai **99,37.** Metode deteksi diintegrasikan metode mitigasi *multi-layer security* untuk semakin meningkat keamanan aplikasi *website* dari serangan SQLi.

Tabel 4.32. Pengujian Performa Metode Deteksi Dataset *Challenge* SQLi

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
1	Gradient Boosting	31160	2567	92,39	0:00:20	20	0,000592997	0,5	TTT
2	Support Vector Machine	31158	2569	92,38	0:00:34	34	0,001008094	0,5	TTT
3	K-Nearest Neigbors	31181	2546	92,45	0:02:16	136	0,004032378	0,5	TTT
4	Logistic Regression	30712	3015	91,06	0:00:11	11	0,000326148	0,5	TTT
5	Naive Bayes	29136	4591	86,39	0:00:04	4	0,000118599	0,5	TTT
6	Support Vector Machine	33237	490	98,55	0:00:22	22	0,000652296	1	TFF
7	K-Nearest Neigbors	33245	482	98,57	0:03:08	188	0,005574169	1	TFF
8	Gradient Boosting	33032	695	97,94	0:00:16	16	0,000474397	1	TFF
9	Support Vector Machine	32990	737	97,81	0:00:20	20	0,000592997	1	TTT
10	K-Nearest Neigbors	33069	658	98,05	0:03:39	219	0,006493314	1	TTT
11	Gradient Boosting	33042	685	97,97	0:00:16	16	0,000474397	1	TTT
12	Logistic Regression	32675	1052	96,88	0:00:10	10	0,000296498	1	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
13	Logistic Regression	31965	1762	94,78	0:00:10	10	0,000296498	1	TTT
14	Naive Bayes	31961	1766	94,76	0:00:04	4	0,000118599	1	TFF
15	Naive Bayes	30636	3091	90,84	0:00:04	4	0,000118599	1	TTT
16	Support Vector Machine	33376	351	98,96	0:00:13	13	0,000385448	1,5	TFF
17	K-Nearest Neigbors	33417	310	99,08	0:04:55	295	0,008746701	1,5	TFF
18	Gradient Boosting	33398	329	99,02	0:00:14	14	0,000415098	1,5	TFF
19	Logistic Regression	33216	511	98,48	0:00:09	9	0,000266849	1,5	TFF
20	Naive Bayes	32403	1324	96,07	0:00:04	4	0,000118599	1,5	TFF
21	K-Nearest Neigbors	33410	317	99,06	0:06:02	362	0,01073324	2	TFF
22	Support Vector Machine	33469	258	99,24	0:00:14	14	0,000415098	2	TFF
23	Gradient Boosting	33450	277	99,18	0:00:17	17	0,000504047	2	TFF
24	Logistic Regression	33323	404	98,80	0:00:11	11	0,000326148	2	TFF
25	Naive Bayes	33000	727	97,84	0:00:06	6	0,000177899	2	TFF
26	Support Vector Machine	33398	329	99,02	0:00:12	12	0,000355798	2,5	TFF
27	K-Nearest Neigbors	33439	288	99,15	0:09:31	571	0,016930056	2,5	TFF
28	Gradient Boosting	33419	308	99,09	0:00:14	14	0,000415098	2,5	TFF
29	Logistic Regression	33361	366	98,91	0:00:09	9	0,000266849	2,5	TFF
30	Naive Bayes	32911	816	97,58	0:00:06	6	0,000177899	2,5	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
31	K-Nearest Neigbors	33448	279	99,17	0:09:51	591	0,017523053	3	TFF
32	Support Vector Machine	33377	350	98,96	0:00:15	15	0,000444748	3	TFF
33	Logistic Regression	33430	297	99,12	0:00:11	11	0,000326148	3	TFF
34	Gradient Boosting	33393	334	99,01	0:00:19	19	0,000563347	3	TFF
35	Naive Bayes	32962	765	97,73	0:00:07	7	0,000207549	3	TFF
36	K-Nearest Neigbors	33416	311	99,08	0:12:31	751	0,022267026	3,5	TFF
37	Support Vector Machine	33503	224	99,34	0:00:13	13	0,000385448	3,5	TFF
38	Gradient Boosting	33383	344	98,98	0:00:15	15	0,000444748	3,5	TFF
39	Logistic Regression	33437	290	99,14	0:00:10	10	0,000296498	3,5	TFF
40	Naive Bayes	32941	786	97,67	0:00:08	8	0,000237199	3,5	TFF
41	K-Nearest Neigbors	33439	288	99,15	0:13:02	782	0,023186171	4	TFF
42	Support Vector Machine	33516	211	99,37	0:00:17	17	0,000504047	4	TFF
43	Gradient Boosting	33421	306	99,09	0:00:16	16	0,000474397	4	TFF
44	Logistic Regression	33441	286	99,15	0:00:11	11	0,000326148	4	TFF
45	Naive Bayes	32953	774	97,71	0:00:08	8	0,000237199	4	TFF
46	Support Vector Machine	33519	208	99,38	0:00:15	15	0,000444748	4,5	TFF
47	K-Nearest Neigbors	33534	193	99,43	0:15:19	919	0,027248199	4,5	TFF
48	Logistic Regression	33462	265	99,21	0:00:10	10	0,000296498	4,5	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
49	Gradient Boosting	33422	305	99,10	0:00:14	14	0,000415098	4,5	TFF
50	Naive Bayes	32953	774	97,71	0:00:09	9	0,000266849	4,5	TFF
51	Support Vector Machine	33539	188	99,44	0:00:20	20	0,000592997	5	TFF
52	K-Nearest Neigbors	33431	296	99,12	0:16:32	992	0,029412637	5	TFF
53	Logistic Regression	33474	253	99,25	0:00:12	12	0,000355798	5	TFF
54	Gradient Boosting	33460	267	99,21	0:00:18	18	0,000533697	5	TFF
55	Naive Bayes	32949	778	97,69	0:00:10	10	0,000296498	5	TFF
56	Support Vector Machine	33519	208	99,38	0:00:18	18	0,000533697	5,5	TFF
57	K-Nearest Neigbors	33536	191	99,43	0:18:17	1097	0,032525869	5,5	TFF
58	Logistic Regression	33499	228	99,32	0:00:10	10	0,000296498	5,5	TFF
59	Gradient Boosting	33452	275	99,18	0:00:14	14	0,000415098	5,5	TFF
60	Naive Bayes	32957	770	97,72	0:00:10	10	0,000296498	5,5	TFF
61	Support Vector Machine	33519	208	99,38	0:00:24	24	0,000711596	6	TFF
62	K-Nearest Neigbors	33400	327	99,03	0:22:03	1323	0,039226732	6	TFF
63	Gradient Boosting	33448	279	99,17	0:00:17	17	0,000504047	6	TFF
64	Logistic Regression	33493	234	99,31	0:00:12	12	0,000355798	6	TFF
65	Naive Bayes	32907	820	97,57	0:00:11	11	0,000326148	6	TFF
66	**Support Vector Machine**	**33514**	**213**	**99,37**	**0:00:34**	**34**	**0,001008094**	**6,5**	**TFF**

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
67	K-Nearest Neigbors	33422	305	99,10	0:31:10	1870	0,055445192	6,5	TFF
68	Logistic Regression	33727	222	100,00	0:00:15	15	0,000444748	6,5	TFF
69	Gradient Boosting	33456	271	99,20	0:00:22	22	0,000652296	6,5	TFF
70	Naive Bayes	32959	768	97,72	0:00:14	14	0,000415098	6,5	TFF
71	Support Vector Machine	33507	220	99,35	0:00:24	24	0,000711596	7	TFF
72	K-Nearest Neigbors	33484	243	99,28	0:30:35	1835	0,054407448	7	TFF
73	Logistic Regression	33492	235	99,30	0:00:12	12	0,000355798	7	TFF
74	Gradient Boosting	33442	285	99,15	0:00:17	17	0,000504047	7	TFF
75	Naive Bayes	32905	822	97,56	0:00:13	13	0,000385448	7	TFF
76	Support Vector Machine	33505	222	99,34	0:00:28	28	0,000830195	7,5	TFF
77	K-Nearest Neigbors	33426	301	99,11	0:32:40	1960	0,058113677	7,5	TFF
78	Logistic Regression	33506	221	99,34	0:00:11	11	0,000326148	7,5	TFF
79	Gradient Boosting	33424	303	99,10	0:00:16	16	0,000474397	7,5	TFF
80	Naive Bayes	32863	864	97,44	0:00:13	13	0,000385448	7,5	TFF
81	K-Nearest Neigbors	33447	280	99,17	0:32:02	1922	0,056986984	8	TFF
82	Support Vector Machine	33531	196	99,42	0:00:23	23	0,000681946	8	TFF
83	Logistic Regression	33506	221	99,34	0:00:12	12	0,000355798	8	TFF
84	Gradient Boosting	33444	283	99,16	0:00:16	16	0,000474397	8	TFF

ID	*Classifier Name*	*Chal. Valid*	*Chal. Invalid*	*Accuracy*	*Chal. ToP*	*Chal. Top Sec.*	*Chal. Top Per Item*	*Margin*	*Corpus Rule*
85	Naive Bayes	32869	858	97,46	0:00:14	14	0,000415098	8	TFF
86	Support Vector Machine	33514	213	99,37	0:00:26	26	0,000770896	9	TFF
87	K-Nearest Neigbors	33390	337	99,00	0:43:16	2596	0,076970973	9	TFF
88	Logistic Regression	33507	220	99,35	0:00:13	13	0,000385448	9	TFF
89	Gradient Boosting	33457	270	99,20	0:00:19	19	0,000563347	9	TFF
90	Naive Bayes	32875	852	97,47	0:00:17	17	0,000504047	9	TFF
91	Support Vector Machine	33521	206	99,39	0:00:27	27	0,000800546	9,5	TFF
92	K-Nearest Neigbors	33423	304	99,10	0:29:46	1786	0,052954606	9,5	TFF
93	Logistic Regression	33501	226	99,33	0:00:13	13	0,000385448	9,5	TFF
94	Gradient Boosting	33451	276	99,18	0:00:18	18	0,000533697	9,5	TFF
95	Naive Bayes	32852	875	97,41	0:00:17	17	0,000504047	9,5	TFF
96	Support Vector Machine	33518	209	99,38	0:00:29	29	0,000859845	10	TFF
97	K-Nearest Neigbors	33389	338	99,00	0:31:32	1892	0,056097489	10	TFF
98	Logistic Regression	33505	222	99,34	0:00:14	14	0,000415098	10	TFF
99	Gradient Boosting	33459	268	99,21	0:00:19	19	0,000563347	10	TFF
100	Naive Bayes	32858	869	97,42	0:00:18	18	0,000533697	10	TFF

4.2.1.3. Teknik Serangan *Remote Code Execution* (RCE)

A. *Data Preparation*

Seperti yang telah dijelaskan pada hasil penelitian, karakteristik dataset RCE ber-beda dengan dataset lainnya karena dikumpulkan dengan metode *crawling*. Setelah proses *crawling* selesai, jumlah baris dataset RCE adalah 48.192. Dari total dataset tersebut, jumlah baris dataset label *payload* adalah 24.095 dan *non-payload* adalah 24.097. Berikut ini adalah *sample* dataset RCE dengan n = 5.

Tabel 4.33. Sample Dataset Serangan RCE

	Sentence	***Label***
46902	content.usatoday.com/topics/topic/Jamie+Gold	0
24201	123tamilgallery.com/kangna-ranaut/	0
1240	/%252e%252e%255c%252e%252e%255c%252e%252e%255c...	1
35873	blog.motionmedia.com/autodesk-2012-product-keys/	0
20112	%f0%80%80%ae%f0%80%80%ae%c0%2f%f0%80%80%ae%f0%	1

B. *Data Preprocessing and Modeling*

Setelah dilakukan *crawling* dan dikompilasi menjadi *data frame*, peneliti melakukan tahapan *data preprocessing*. Secara umum, pada *data preprocessing*, peneliti melakukan proses *cleaning* dan *positioning*. Pada proses *data cleaning*, terdapat beberapa perlakuan yang dilakukan yaitu (1) mengeliminasi baris dataset yang tidak lengkap atau kosong dan (2) menghapus dataset yang duplikat. Setelah tahapan *data cleaning*, tahapan *data positioning* yang dilakukan adalah (1) melakukan konversi baris data kolom label menjadi data numerik dan (2) memverifikasi tipe *encoding* dataset yang berpotensi menyebabkan *error*.

Setelah melewati tahapan *data cleaning* dan *positioning*, jumlah baris dataset yang *eligible* digunakan untuk penelitian adalah 47.177 baris dataset. Artinya, terdapat 1015 baris dataset yang dieliminasi. Dataset yang *eligible* tersebut lalu dikemas menjadi *corpus* agar pemrosesan teks dapat dilakukan oleh Python NLTK. Saat data telah dikemas menjadi *corpus*, baris dataset label *payload* menjadi 23.080 dan baris dataset label *non-payload* menjadi 24.097 baris dataset.

Margin teroptimal untuk metode deteksi serangan RCE berada pada angka 15, sehingga batas vektor menjadi 365. Artinya, terdapat 365 *identifier* yang digunakan pada label *payload* dan *non-payload*. Untuk menghindari adanya *missing link* dalam pembahasan metode deteksi serangan RCE, berikut ini ditampilkan 100 dari 365 vektor yang digunakan pada serangan SQLi label *payload* dan *non-payload*.

Tabel 4.34. Daftar 100 dari 365 vektor Serangan RCE *Label Payload*

No	Vektor	Jumlah Kemunculan
1	%	323227
2	80	70560
3	c0	53850
4	ae	25104
5	%%	24883
6	c1	16944
7	2e	16543
8	.%	13368
9	25c0	9120
10	32	9118
11	f0	8400
12	f8	8400
13	e0	8400
14	af	8262
15	..%	7417

No	Vektor	Jumlah Kemunculan
16	5c	6311
17	.	6294
18	65	5459
19	252e	5448
20	uff0e	5448
21	25ae	5448
22	ini	5313
23	fc	5040
24	6e	4900
25	5e	4872
26	ee	4872
27	fe	4872
28	2f	3805
29	35	3551
30	25c1	3504
31	bg	3360
32	qf	3240
33	0	2591
34	afetc	2520
35	/	2379
36	1	2352
37	\	2082
38	.?%	2016
39	??%	2016
40	?.%	2016
41	.../.%	2016
42	..../%	2016
43	63	1829
44	66	1794
45	etc	1784
46	25af	1764
47	u2215	1764
48	u2216	1764
49	252f	1764

No	Vektor	Jumlah Kemunculan
50	255c	1764
51	9c	1764
52	259c	1764
53	uefc8	1692
54	uf025	1692
55	pc	1620
56	9v	1620
57	8s	1620
58	1c	1620
59	/%	1358
60	}	1290
61	file	1288
62	\%	1116
63	2fetc	1026
64	5cetc	1026
65	qfetc	1008
66	0x2e0x2e	936
67	afpasswd	840
68	afissue	840
69	afboot	840
70	afwindows	840
71	afsystem32	840
72	afdrivers	840
73	afhosts	840
74	0x2f	672
75	0x5c	672
76	passwd	609
77	{	604
78	boot	590
79	windows	586
80	system32	577
81	hosts	574
82	issue	570
83	drivers	570

No	Vektor	Jumlah Kemunculan
84	25afetc	504
85	252fetc	504
86	255cetc	504
87	9cetc	504
88	pcetc	504
89	9vetc	504
90	8setc	504
91	1cetc	504
92	u2215etc	504
93	u2216etc	504
94	uefc8etc	504
95	uf025etc	504
96	66etc	504
97	63etc	504
98	259cetc	504
99	.%%	504
100	..	465

Berdasarkan tabel vektor *payload* dan tinjauan terhadap vektor-vektor yang muncul, dataset RCE label *payload* terdiri dari kode-kode serangan RCE untuk beberapa *platform* yang didominasi oleh sistem operasi Linux dan Windows. Dominasi tersebut terjadi karena kebanyakan *web server* saat ini berjalan di atas Linux dan Windows. Oleh karena itu, kode-kode serangan RCE pada tabel 4.35 tersebut juga didominasi oleh *built in command* sistem operasi Linux dan Windows, seperti *Bash* untuk Linux dan *Power Script* untuk Windows. Selain itu, terdapat juga kode serangan RCE berbasis bahasa pemrograman *scripting* seperti PHP, Ruby, Python, dan lain-lain. Kode-kode RCE tersebut berupaya mengeksploitasi fungsi-fungsi atau *built-in functions* dari bahasa pemrograman. Sementara itu, berikut ini adalah kode serangan RCE dengan label *non-payload*.

Tabel 4.35. Daftar 100 dari 365 vektor Serangan RCE Label *Non-Payload*

No	Vektor	Jumlah Kemunculan
1	/	56071
2	.	49349
3	-	43562
4	com	19399
5	=	5642
6	html	3837
7	?	3128
8	&	2101
9	org	2069
10	ca	2064
11	2011	2011
12	%	1778
13	+	1640
14	index	1579
15	blogspot	1533
16	yahoo	1359
17	articles	1194
18	answers	1071
19	php	1043
20	question	1036
21	qid	1022
22	linkedin	1010
23	in	1000
24	htm	994
25	the	933
26	blog	903
27	2010	854
28	blogs	701
29	pub	700
30	net	681
31	edu	668
32	10	642

No	Vektor	Jumlah Kemunculan
33	aspx	615
34	of	605
35	ancestry	604
36	11	583
37	tag	568
38	wiki	565
39	id	541
40	2009	520
41	news	518
42	content	511
43	and	500
44	about	497
45	1	493
46	/?	465
47	a	447
48	8	445
49	9	443
50	p	442
51	wordpress	420
52	boards	400
53	to	399
54	bleacherreport	391
55	6	385
56	5	378
57	7	363
58	usatoday	354
59	community	342
60	4	333
61	12	325
62	3	324
63	s	315
64	topic	313
65	connect	313
66	:	310

No	Vektor	Jumlah Kemunculan
67	category	308
68	2008	307
69	2	306
70	sports	301
71	1	298
72	acronyms	293
73	thefreedictionary	292
74	for	289
75	article	289
76	2	285
77	sfgate	265
78	commons	264
79	th	262
80	wikimedia	262
81	montreal	261
82	rootsweb	261
83	gov	259
84	wikia	259
85	au	258
86	view	255
87	archiver	255
88	surnames	241
89	nhl	237
90	read	235
91	on	234
92	3d	233
93	profile	223
94	od	223
95	books	211
96	tags	209
97	req	205
98	2007	204
99	mb	199
100	football	193

Berdasarkan tinjauan terhadap daftar vektor label *non-payload*, vektor *non-payload* didominasi oleh *normal URL pattern* atau pola-pola penulisan URL yang normal dan *safe*. Hal ini terjadi karena kebanyakan serangan RCE disisipkan pada URL dan dilakukan dengan metode GET atau POST. Oleh karena itu, metode deteksi harus mampu membedakan *vulnerable URL* dan *unvulnerable URL*. Metode deteksi RCE harus mampu meningkatkan jumlah TP dan TN dalam proses belajarnya. Dengan demikian, metode deteksi dapat berjalan secara optimal. Sementara itu, vektor-vektor yang saling beririsan adalah sebagai berikut.

Tabel 4.36. Irisan Vektor Label Payload dan Non-Payload RCE

No	Vektor	Payload	Non-Payload	Degree to Payload	Tendency
1	%	323227	1778	321449	payload
2	80	70560	64	70496	payload
3	.	6294	49349	-43055	non-payload
4	/	2379	56071	-53692	non-payload
5	1	2352	298	2054	payload
6	file	1288	110	1178	payload
7	-	153	43562	-43409	non-payload
8	html	99	3837	-3738	non-payload
9	index	97	1579	-1482	non-payload
10	htm	96	994	-898	non-payload
11	bin	94	54	40	payload
12	c	81	148	-67	non-payload
13	&	66	2101	-2035	non-payload
14	id	64	541	-477	non-payload
15	(	56	74	-18	non-payload
16	25	55	121	-66	non-payload
17	=	50	5642	-5592	non-payload

No	Vektor	Payload	Non-Payload	Degree to Payload	Tendency
18	?	46	3128	-3082	non-payload
19	dir	44	143	-99	non-payload
20	com	41	19399	-19358	non-payload
21	php	39	1043	-1004	non-payload
22	n	31	122	-91	non-payload

C. *Feature Enginering, Data Training, and Data Testing*

Sama halnya dengan serangan-serangan sebelumnya, *feature enginering* pada serangan RCE dilakukan dalam dua tahapan, yaitu (1) kontruksi corpus dan (2) kontruksi fitur. Kontruksi corpus dilakukan agar dataset diolah berdasarkan aturan *corpus* yang konsisten. Kontruksi fitur digunakan untuk memverifikasi eksistensi atau keberadaan suatu *identifier*. Tahapan kontruksi corpus, beserta *coding*-nya dalam bahasa pemrograman Python dapat dilihat pada gambar 4.27 dan 4.28. Sementara itu, rincian kode kontruksi fitur dapat dilihat pada gambar 4.29.

Setelah dilakukan tahapan optimalisasi kinerja metode, ukuran data uji teroptimal berada pada pengaturan 0,2 *test size*. Penentuan ukuran data uji tersebut didasarkan pada temuan tingkat akurasi pada tahapan optimalisasi. Berdasarkan proses optimalisasi, ukuran 0,2 *test size* dapat lebih komprehensif dan representatif dengan kebutuhan metode deteksi. Ukuran data uji ini sama dengan yang digunakan pada serangan XSS dan RCE. Sebanyak 80% dataset digunakan untuk *data training* dan sebanyak 20% digunakan untuk *data testing*. Setelah mendapatkan tingkat akurasi, peneliti juga melakukan *confussion matrix* untuk mendapatkan informasi TP, TN, FP, dan FN, serta data tentang *precision, recall,* dan nilai f-1.

D. *Performance Optimalization*

Untuk mendapatkan kinerja metode deteksi RCE yang paling akurat, optimalisasi kinerja metode deteksi RCE juga dilakukan secara komprehensif dan sistematis. Peneliti mengujikan lima algoritma, untuk mencari algoritma yang mampu mendapatkan tingkat akurasi tertinggi dengan ToP yang relatif rendah. Selain pengujian pada lima algoritma, peneliti juga melakukan optimalisasi konfigurasi. Optimalisasi konfigurasi ini dilakukan untuk semakin meningkatkan tingkat akurasi. Rekam jejak upaya optimalisasi algoritma dan konfigurasi parameter dapat dilihat pada tabel di bawah ini.

Tabel 4.37. Tahapan Optimalisasi Kinerja Metode Deteksi RCE

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
1	**Support Vector Machine**	**0,9874947**	**0:01:21**	**15**	**365**	**9436**	**37741**	**True**	**False**	**False**
2	K-Nearest Neigbors	0,98537516	0:08:19	15	365	9436	37741	True	False	False
3	Support Vector Machine	0,98495125	0:00:53	7	170	9436	37741	True	False	False
4	Support Vector Machine	0,9847393	0:01:02	9	219	9436	37741	True	False	False
5	Support Vector Machine	0,98431539	0:01:11	10	243	4718	42459	True	False	False
6	K-Nearest Neigbors	0,98431539	0:04:04	10	243	4718	42459	True	False	False
7	Logistic Regression	0,98420941	0:00:24	15	365	9436	37741	True	False	False
8	K-Nearest Neigbors	0,98399746	0:05:31	7	170	9436	37741	True	False	False

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
9	Support Vector Machine	0,9837855	0:00:55	8	195	9436	37741	True	False	False
10	K-Nearest Neigbors	0,98325562	0:05:53	8	195	9436	37741	True	False	False
11	Logistic Regression	0,98325562	0:00:11	7	170	9436	37741	True	False	False
12	Support Vector Machine	0,9824078	0:00:39	6	146	9436	37741	True	False	False
13	K-Nearest Neigbors	0,98230182	0:04:26	6	146	9436	37741	True	False	False
14	Logistic Regression	0,98219585	0:00:14	9	219	9436	37741	True	False	False
15	K-Nearest Neigbors	0,98219585	0:06:35	9	219	9436	37741	True	False	False
16	Logistic Regression	0,98155998	0:00:17	10	243	4718	42459	True	False	False
17	Gradient Boosting	0,98155998	0:00:40	10	243	4718	42459	True	False	False
18	Gradient Boosting	0,98124205	0:00:30	7	170	9436	37741	True	False	False
19	Gradient Boosting	0,98124205	0:00:38	9	219	9436	37741	True	False	False
20	Logistic Regression	0,98113607	0:00:13	8	195	9436	37741	True	False	False
21	Gradient Boosting	0,98060619	0:01:01	15	365	9436	37741	True	False	False
22	Gradient Boosting	0,98039423	0:00:33	8	195	9436	37741	True	False	False
23	K-Nearest Neigbors	0,9800763	0:03:37	5	121	9436	37741	False	False	False
24	Support Vector Machine	0,97997033	0:00:39	5	121	9436	37741	False	False	False
25	Logistic Regression	0,9796524	0:00:10	6	146	9436	37741	True	False	False
26	Gradient Boosting	0,9786986	0:00:23	6	146	9436	37741	True	False	False

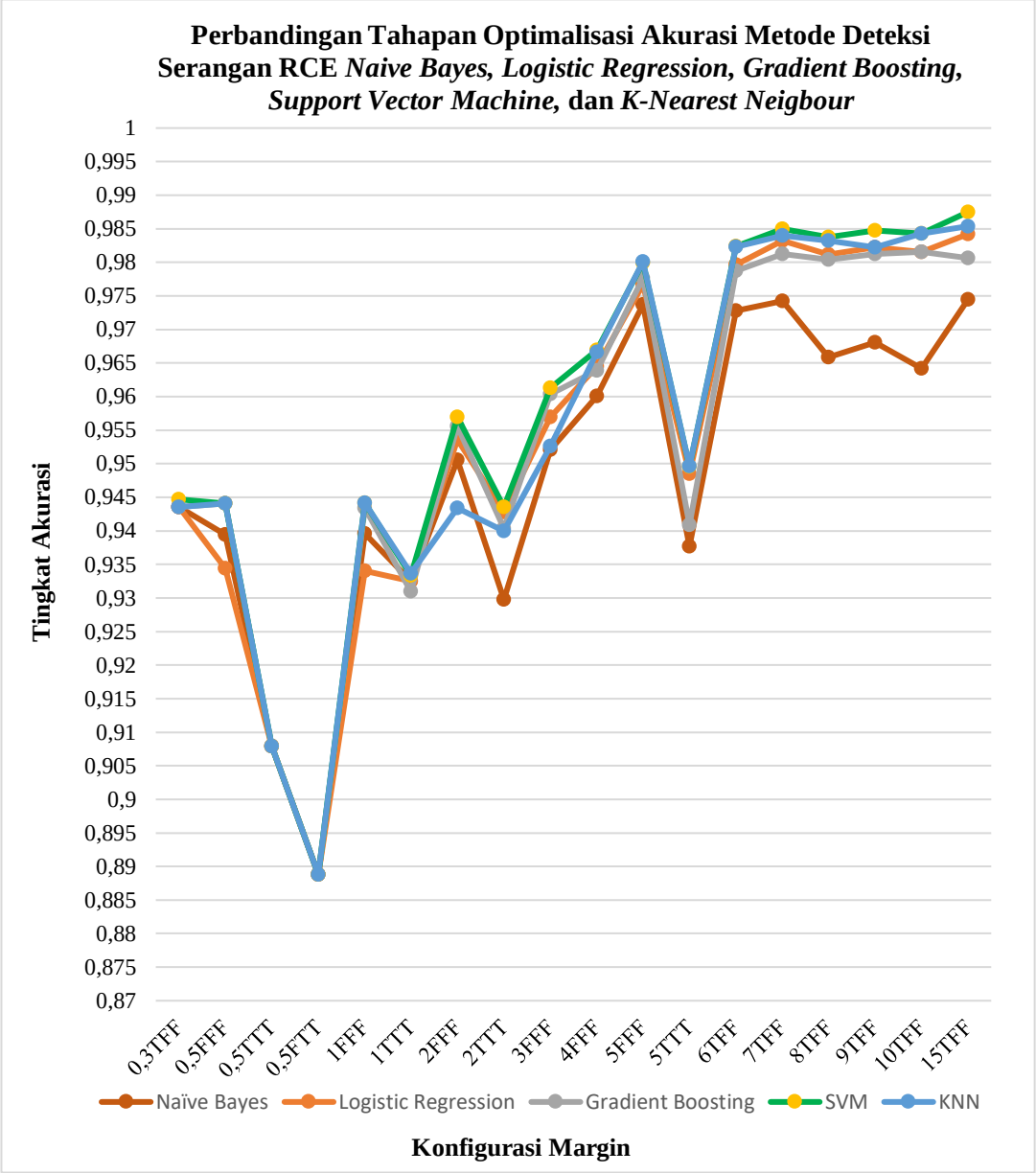
	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
27	Gradient Boosting	0,97763883	0:00:19	5	121	9436	37741	False	False	False
28	Logistic Regression	0,97668504	0:00:07	5	121	9436	37741	False	False	False
29	Naive Bayes	0,97445952	0:00:24	15	365	9436	37741	True	False	False
30	Naive Bayes	0,97424756	0:00:10	7	170	9436	37741	True	False	False
31	Naive Bayes	0,97371768	0:00:07	5	121	9436	37741	False	False	False
32	Naive Bayes	0,97276388	0:00:09	6	146	9436	37741	True	False	False
33	Naive Bayes	0,96810089	0:00:14	9	219	9436	37741	True	False	False
34	Support Vector Machine	0,96693514	0:00:40	4	97	9436	37741	False	False	False
35	K-Nearest Neigbors	0,96661721	0:02:49	4	97	9436	37741	False	False	False
36	Naive Bayes	0,96587537	0:00:13	8	195	9436	37741	True	False	False
37	Logistic Regression	0,96449767	0:00:06	4	97	9436	37741	False	False	False
38	Naive Bayes	0,96417974	0:00:13	10	243	4718	42459	True	False	False
39	Gradient Boosting	0,96386181	0:00:14	4	97	9436	37741	False	False	False
40	Support Vector Machine	0,96130304	0:00:52	3	72	9639	38553	False	False	False
41	Gradient Boosting	0,96036933	0:00:11	3	72	9639	38553	False	False	False
42	Naive Bayes	0,96004663	0:00:06	4	97	9436	37741	False	False	False
43	Logistic Regression	0,95694574	0:00:05	3	72	9639	38553	False	False	False
44	Support Vector Machine	0,95694574	0:00:41	2	48	9639	38553	False	False	False

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
45	Gradient Boosting	0,95559705	0:00:10	2	48	9639	38553	False	False	False
46	Logistic Regression	0,95362589	0:00:03	2	48	9639	38553	False	False	False
47	K-Nearest Neigbors	0,95258844	0:02:11	3	72	9639	38553	False	False	False
48	Naive Bayes	0,95206972	0:00:05	3	72	9639	38553	False	False	False
49	Naive Bayes	0,95051354	0:00:03	2	48	9639	38553	False	False	False
50	K-Nearest Neigbors	0,94966087	0:03:22	5	121	9436	37741	True	True	True
51	Support Vector Machine	0,9495549	0:01:29	5	121	9436	37741	True	True	True
52	Logistic Regression	0,94849513	0:00:08	5	121	9436	37741	True	True	True
53	Support Vector Machine	0,94467995	0:00:15	0.3	7	9436	37741	True	False	False
54	Support Vector Machine	0,94418508	0:00:36	1	24	9639	38553	False	False	False
55	K-Nearest Neigbors	0,94418508	0:01:08	1	24	9639	38553	False	False	False
56	Gradient Boosting	0,94408134	0:00:03	0.5	12	9639	38553	False	False	False
57	Support Vector Machine	0,94408134	0:00:23	0.5	12	9639	38553	False	False	False
58	K-Nearest Neigbors	0,94408134	0:00:40	0.5	12	9639	38553	False	False	False
59	Support Vector Machine	0,9435142	0:00:41	2	48	9436	37741	True	True	True
60	Naive Bayes	0,9435142	0:00:00	0.3	7	9436	37741	True	False	False
61	Logistic Regression	0,9435142	0:00:00	0.3	7	9436	37741	True	False	False
62	Gradient Boosting	0,9435142	0:00:03	0.3	7	9436	37741	True	False	False

	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
63	K-Nearest Neigbors	0,9435142	0:00:34	0.3	7	9436	37741	True	False	False
64	K-Nearest Neigbors	0,94340822	0:01:25	2	48	9436	37741	True	True	True
65	Gradient Boosting	0,94335512	0:00:05	1	24	9639	38553	False	False	False
66	Logistic Regression	0,94277236	0:00:03	2	48	9436	37741	True	True	True
67	Gradient Boosting	0,94086477	0:00:18	5	121	9436	37741	True	True	True
68	Gradient Boosting	0,94033489	0:00:07	2	48	9436	37741	True	True	True
69	K-Nearest Neigbors	0,93993153	0:01:55	2	48	9639	38553	False	False	False
70	Naive Bayes	0,93962029	0:00:01	1	24	9639	38553	False	False	False
71	Naive Bayes	0,9394128	0:00:01	0.5	12	9639	38553	False	False	False
72	Naive Bayes	0,93768546	0:00:07	5	121	9436	37741	True	True	True
73	Logistic Regression	0,93443303	0:00:01	0.5	12	9639	38553	False	False	False
74	Logistic Regression	0,93401805	0:00:01	1	24	9639	38553	False	False	False
75	K-Nearest Neigbors	0,93365833	0:00:46	1	24	9436	37741	True	True	True
76	Support Vector Machine	0,9333404	0:00:25	1	24	9436	37741	True	True	True
77	Naive Bayes	0,93249258	0:00:01	1	24	9436	37741	True	True	True
78	Logistic Regression	0,93249258	0:00:01	1	24	9436	37741	True	True	True
79	Gradient Boosting	0,9310089	0:00:03	1	24	9436	37741	True	True	True
80	Naive Bayes	0,92973718	0:00:03	2	48	9436	37741	True	True	True

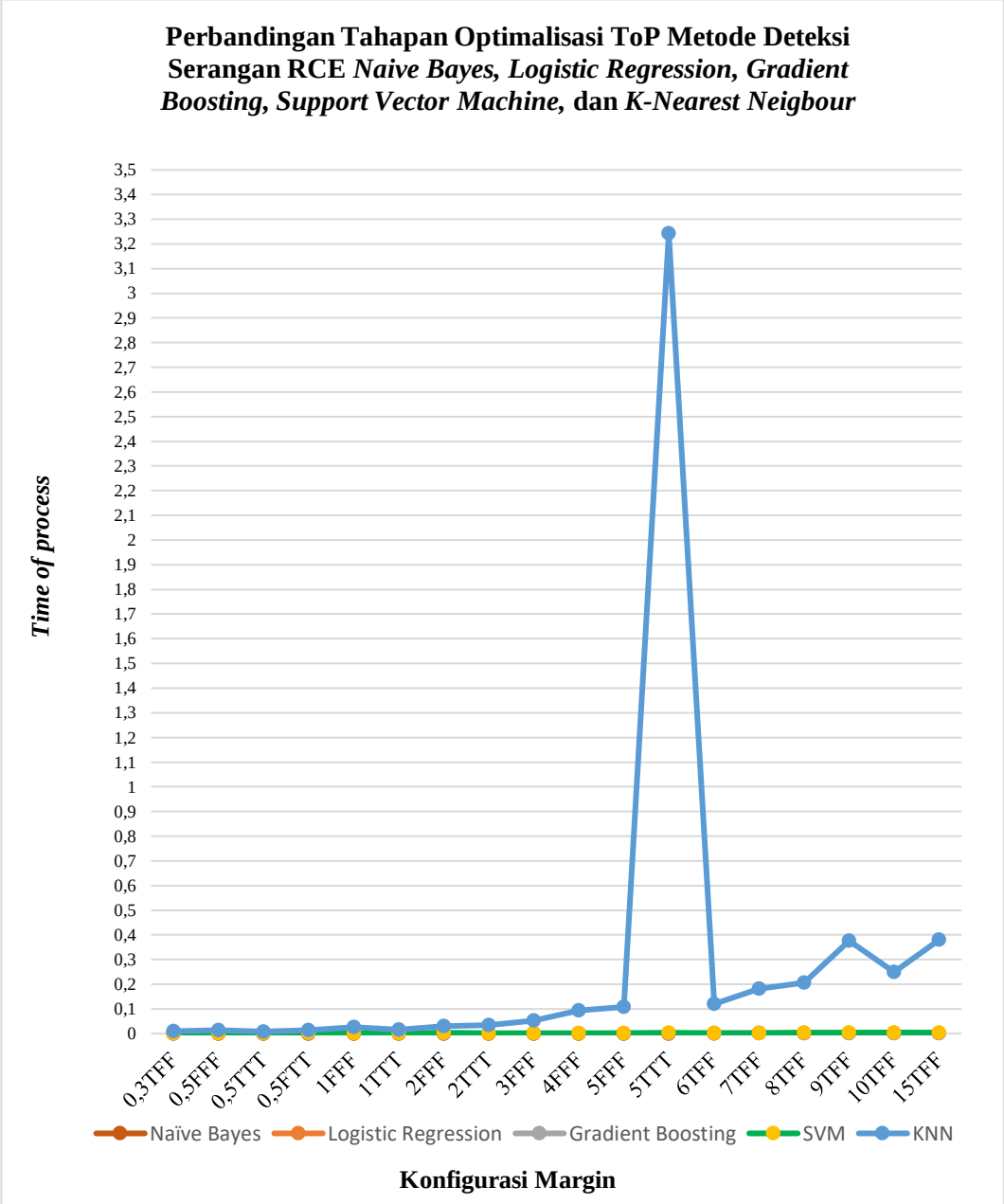
	Classifier Name	*Accuracy*	*ToP*	*Margin*	*Vector Limit*	*Total Test*	*Total Train*	*Lowercase*	*Alphanumeric*	*Remove Puncts*
81	Naive Bayes	0,90790589	0:00:00	0.5	12	9436	37741	True	True	True
82	Logistic Regression	0,90790589	0:00:01	0.5	12	9436	37741	True	True	True
83	Gradient Boosting	0,90790589	0:00:02	0.5	12	9436	37741	True	True	True
84	Support Vector Machine	0,90790589	0:00:22	0.5	12	9436	37741	True	True	True
85	K-Nearest Neigbors	0,90790589	0:00:26	0.5	12	9436	37741	True	True	True
86	Naive Bayes	0,88878514	0:00:01	0.5	12	9639	38553	False	True	False
87	Logistic Regression	0,88878514	0:00:01	0.5	12	9639	38553	False	True	False
88	Gradient Boosting	0,88878514	0:00:04	0.5	12	9639	38553	False	True	False
89	Support Vector Machine	0,88878514	0:00:46	0.5	12	9639	38553	False	True	False
90	K-Nearest Neigbors	0,88878514	0:00:38	0.5	12	9639	38553	False	True	False

Berdasarkan hasil tahapan optimalisasi yang dilakukan, SVM menjadi pilihan algoritma untuk mendeteksi serangan RCE. SVM mampu mendapatkan tingkat akurasi sebesar 0,9874947 secara konsisten. Sementara itu, tingkat akurasi terendah diperoleh oleh K-Nearest Neigbor dengan tingkat akurasi sebesar 0,88878514. Temuan ini sekaligus menunjukan bahwa metode SVM secara konsisten terpilih untuk semua teknik serangan, mulai dari XSS, SQLi, dan RCE. Visualisasi perbandingan tahapan optimalisasi akurasi metode deteksi serangan RCE dapat dilihat pada grafik-grafik berikut ini.



Grafik 4.36. Perbandingan Tahapan Optimalisasi Akurasi Metode Deteksi RCE

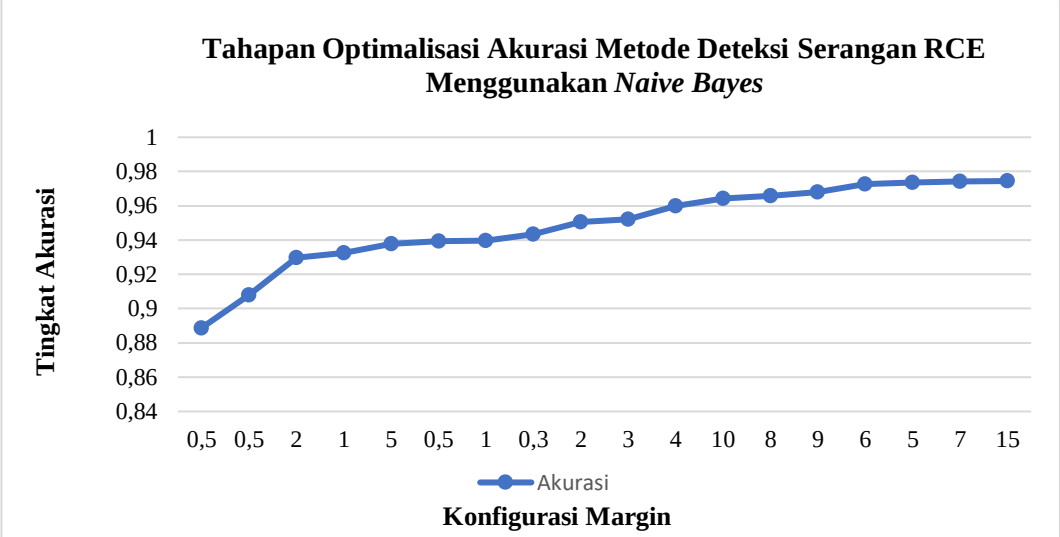
Berdasarkan grafik perbandingan tahapan optimalisasi tersebut, kesenjangan tingkat akurasi tampak pada Naïve Bayes. Ketika *margin* berada pada posisi 6, 7, 8, 9, 10, dan 15, tingkat akurasi Naïve Bayes semakin tidak stabil dan dominan terjadi penurun. Terkait dengan penurunan tingkat akurasi pada margin 0,5 yang ketiga, 1 yang kedua, dan 5 yang kedua, hal tersebut terjadi karena adanya perubahan aturan corpus (lihat tabel 4.38 dengan pengaturan corpus True, True, True.



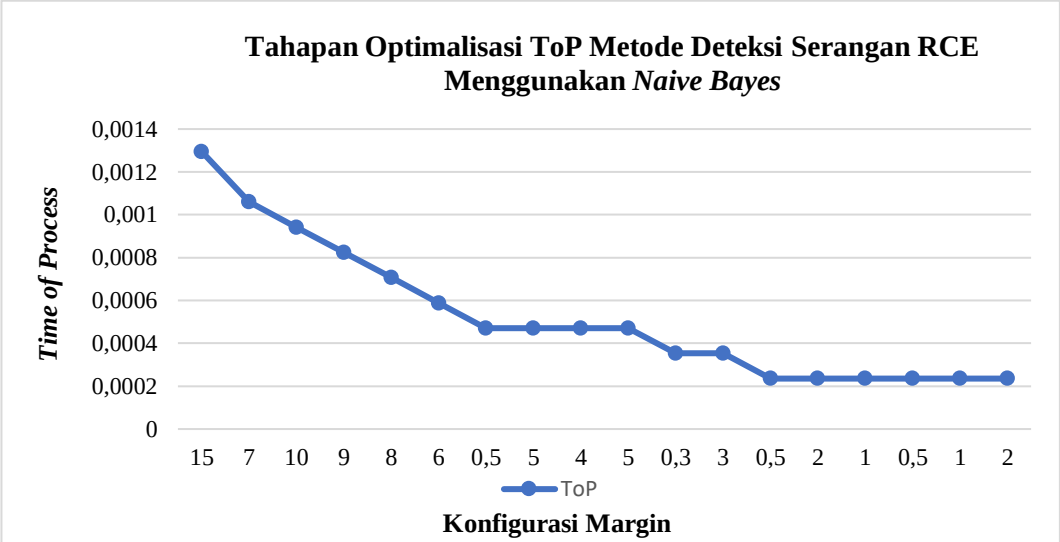
Grafik 4.37. Perbandingan Tahapan Optimalisasi ToP Metode Deteksi RCE

Berdasarkan data ToP, KNN secara konsisten menjadi algoritma yang memerlukan ToP lebih tinggi dibandingkan yang lainnya. Bahkan, pada posisi margin 5, KNN memerlukan waktu 3,24 detik untuk mendeteksi satu *input*. Temuan ini sekaligus menunjukan bahwa KNN memerlukan durasi ToP pada semua teknik serangan, baik pada serangan XSS, SQLi, maupun RCE.

1) Tingkat Akurasi dan ToP *Naïve Bayes*



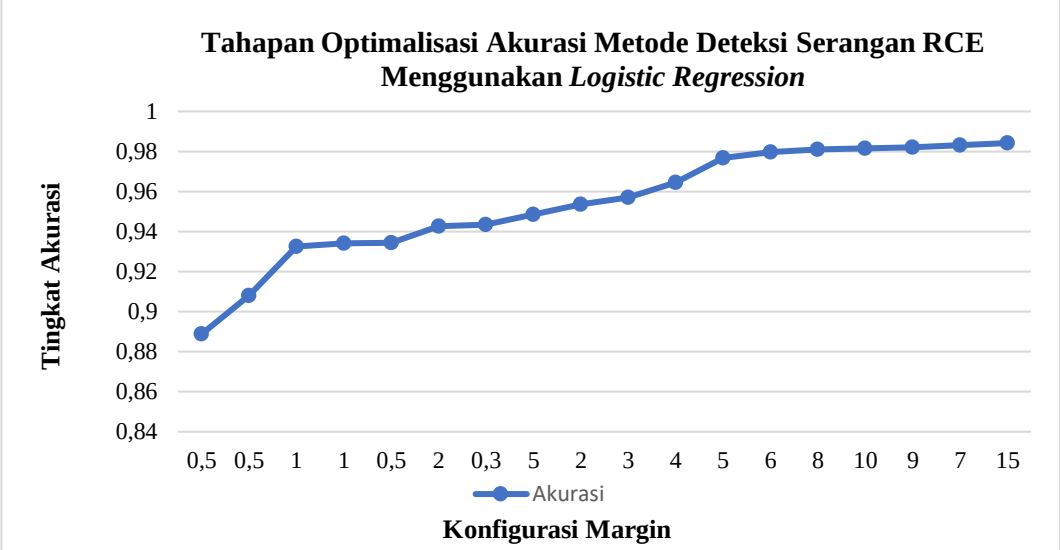
Grafik 4.38. Tahapan Optimalisasi Akurasi Metode Deteksi RCE *Naïve Bayes*



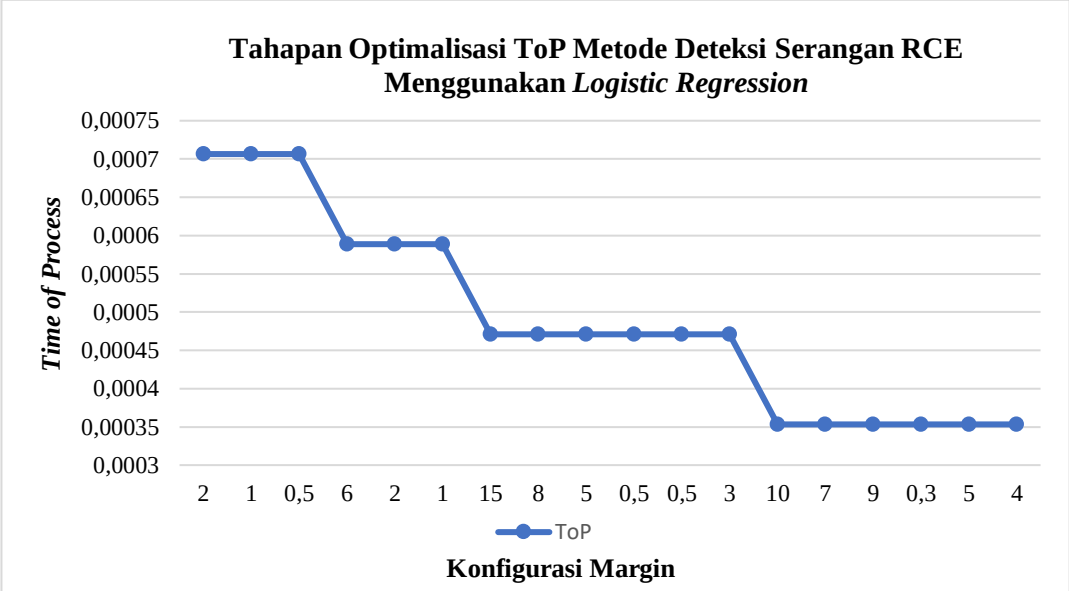
Grafik 4.39. Tahapan Optimalisasi ToP Metode Deteksi RCE *Naïve Bayes*

Tingkat akurasi tertinggi *Naïve Bayes* adalah 0,9744 dengan margin 15. Pada posisi *margin* tersebut, Naïve Bayes membutuhkan ToP sebesar 0,001 mikro detik. Sementara itu, tingkat akurasi terendah Naïve Bayes adalah 0,8887 dengan pengaturan margin sebesar 0,5 dan memerlukan ToP sebesar 0,0002. Tingkat akurasi Naïve Bayes memang tinggi, namun tren optimalisasinya kurang stabil (grafik 4.56)

2) Tingkat Akurasi dan ToP *Logistic Regression*



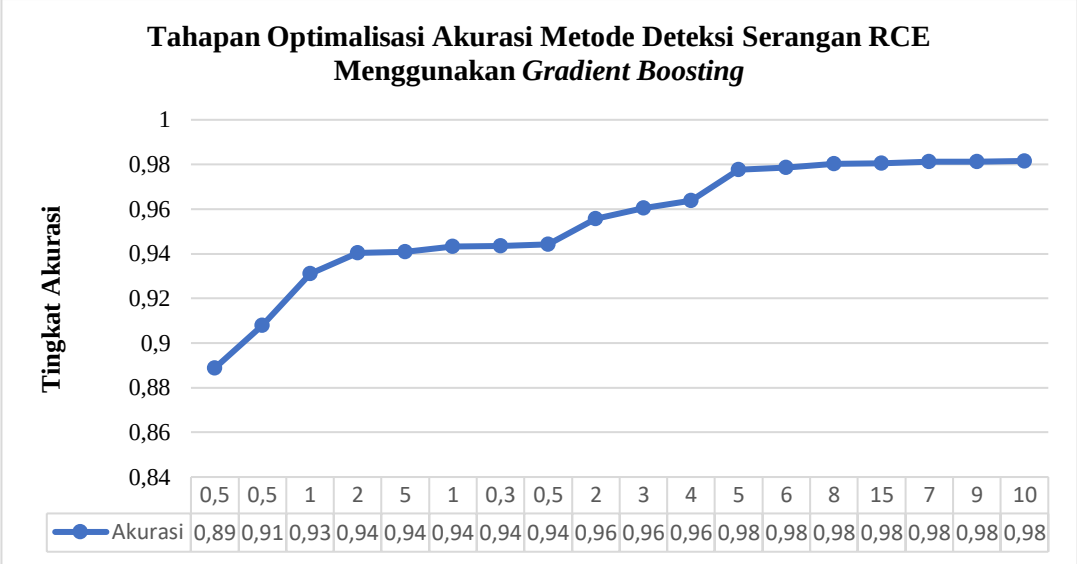
Grafik 4.40. Tahapan Optimalisasi Akurasi Metode Deteksi RCE *Logistic Regression*



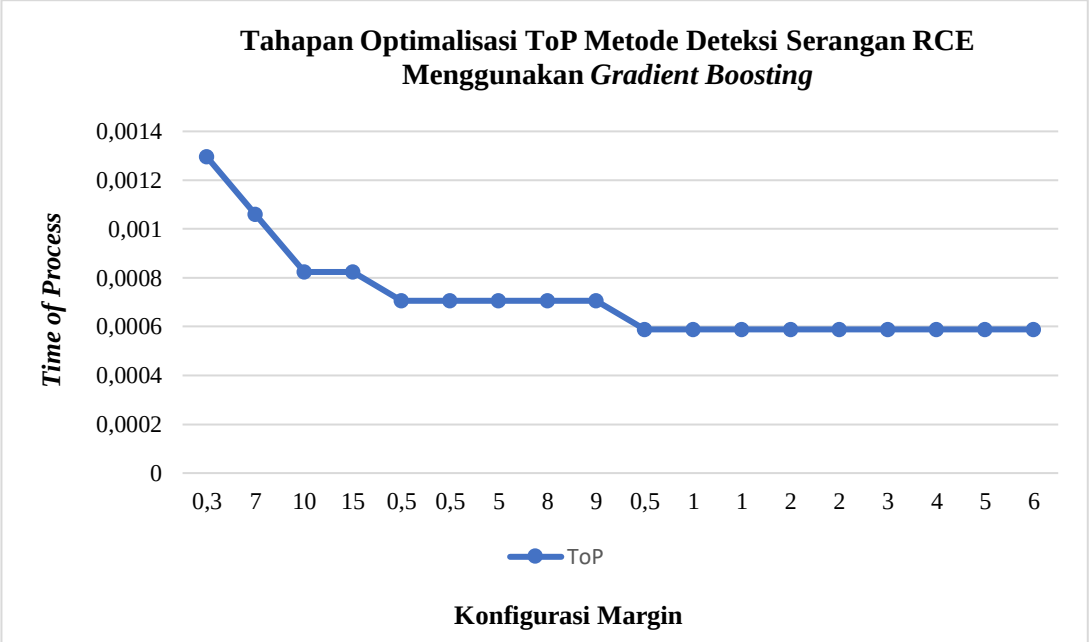
Grafik 4.41.. Tahapan Optimalisasi ToP Metode Deteksi RCE *Logistic Regression*

Tingkat akurasi tertinggi *Logistic Regression* mencapai angka 0,9842 dengan pengaturan *margin* 15. Dengan margin tersebut, *Logistic Regression* memerlukan ToP 0,0004 mikro detik untuk mendeteksi *input*. Tingkat akurasi terendah algoritma ini berada di angka 0,8887 dengan posisi *margin* 0,5 dan ToP 0,0007. Sementara itu, temuan pada tahapan optimalisasi *Gradient Boosting* adalah berikut.

3) Tingkat Akurasi dan ToP *Gradient Boosting*



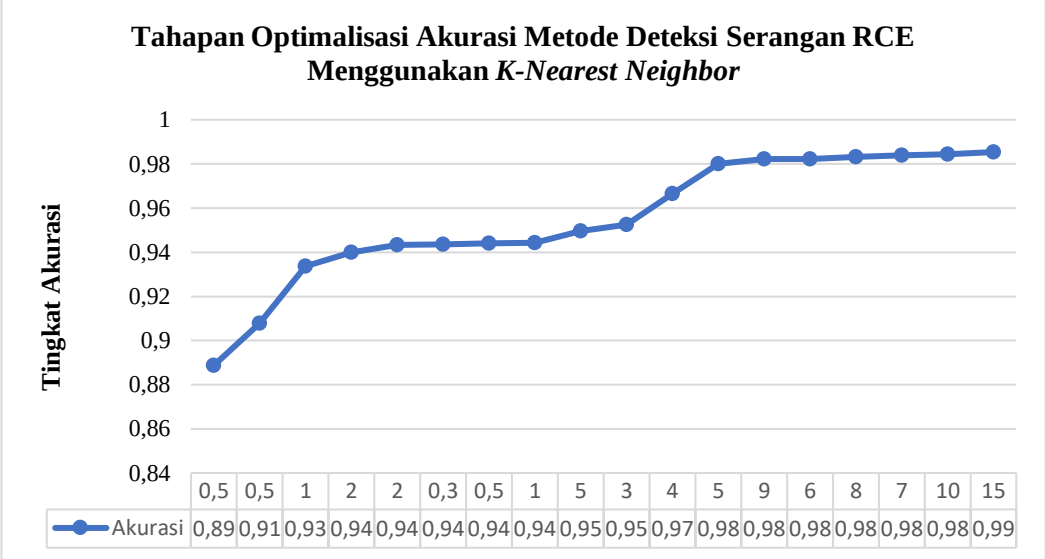
Grafik 4.42.. Tahapan Optimalisasi Akurasi Metode Deteksi RCE *Gradient Bossting*



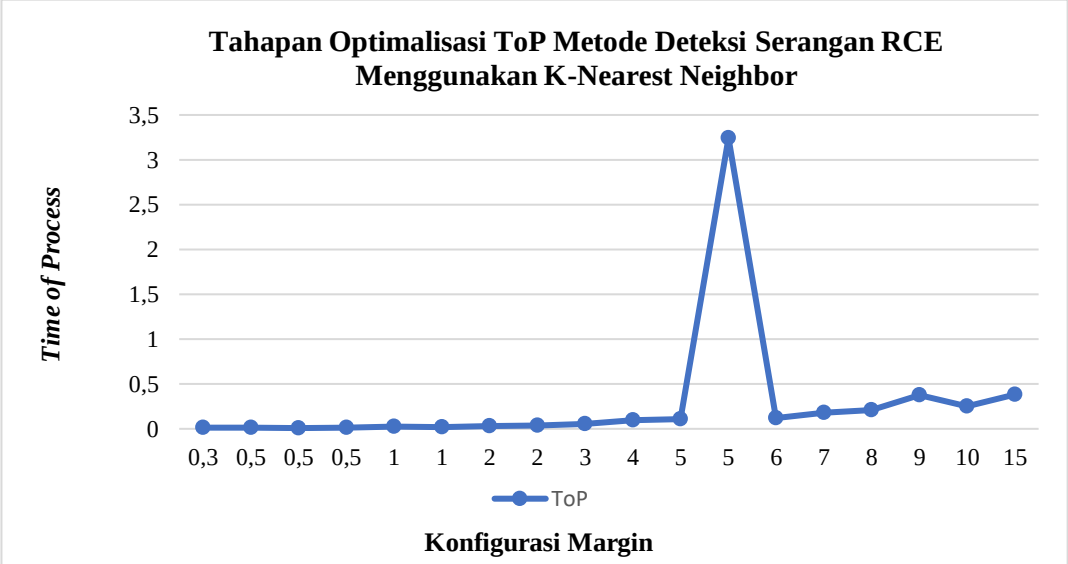
Grafik 4.43.. Tahapan Optimalisasi Akurasi Metode Deteksi RCE *Gradient Boosting*

Tingkat akurasi tertinggi yang mampu dicapai *Gradient Boosting* mencapai angka 0,9815 dengan margin 10 dan ToP sebesar 0,0008. Tingkat akurasi terendah berada pada posisi 0,8887 dengan ToP sebesar 0,0007. Pada konfigurasi teroptimal, algoritma ini mendapatkan tingkat akurasi sebesar 0,9806.

4) Tingkat Akurasi dan ToP *K-Nearest Neighbor*



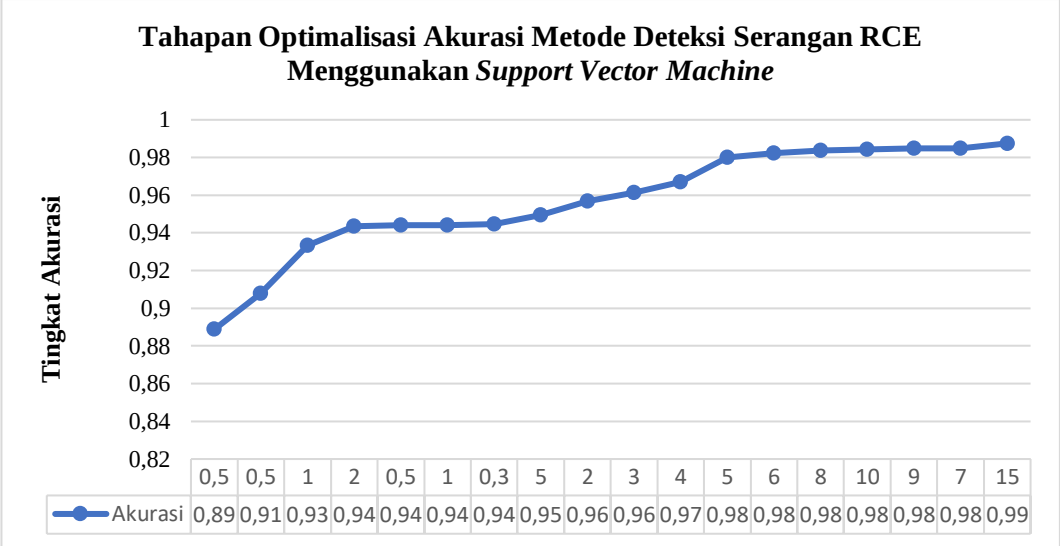
Grafik 4.44. Tahapan Optimalisasi Akurasi Metode Deteksi RCE KNN



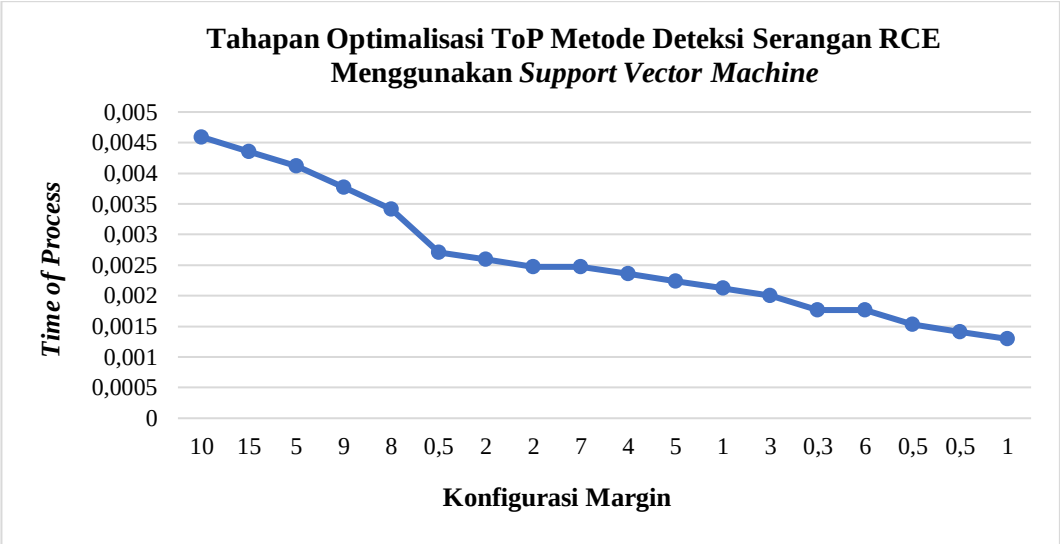
Grafik 4.45. Tahapan Optimalisasi ToP Metode Deteksi RCE KNN

Hasil optimalisasi K-Nearest Neighbor menunjukan bahwa konfigurasi yang paling optimal berada pada *margin* 15 dengan tingkat akurasi 0,9853 dan ToP 0,38. Sementara itu, tingkat akurasi terendah berada pada angka 0,8887 dengan *margin 0,5*. Terkait dengan data ToP, K-Nearest Neigbor memerlukan waktu sampai 3,24 saat parameter konfigurasi berada pada *margin* 5.

5) Tingkat Akurasi dan ToP *Support Vector Machine*



Grafik 4.46. Tahapan Optimalisasi Akurasi Metode Deteksi RCE SVM

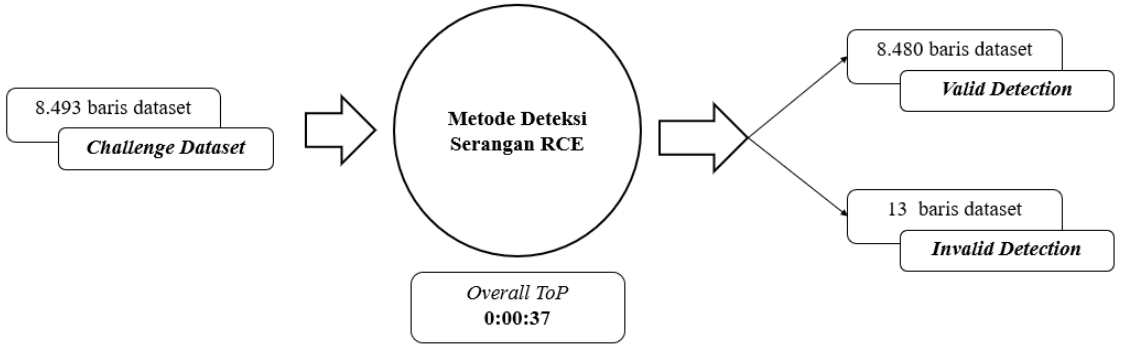


Grafik 4.47. Tahapan Optimalisasi Akurasi Metode Deteksi RCE SVM

SVM menjadi algoritma dengan tingkat akurasi tertinggi dalam hal implementasi deteksi serangan RCE. Dalam tahapan optimalisasi, algoritma ini mampu mencapai tingkat akurasi 0,9874 dengan *margin* 15 dan ToP 0,004. Tingkat akurasi terendah yang didapat SVM adalah 0,8887 dengan *margin* 0,5 dan ToP 0,001. Dengan tingkat akurasi ini, SVM menjadi algoritma terpilih dalam deteksi RCE.

E. *Performance Testing*

Pengujian kinerja metode deteksi serangan XSS juga dilakukan untuk melihat kinerja algoritma terpilih dalam mendeteksi serangan RCE. Dataset yang digunakan untuk pengujian kinerja adalah dataset yang berbeda dengan yang digunakan pada proses *data training* dan *testing*. Total baris dataset yang diujikan dalam tahapan ini adalah 8.493 baris dataset. Hasil pengujian kinerja metode deteksi RCE menggunakan SVM dapat dilihat pada ID 86 tabel di bawah ini. Berdasarkan tabel tersebut, SVM mampu mendeteksi **99,85%** label baris dataset secara valid, dengan rincian 8.480 *valid* dan 13 *invalid prediction*.



Gambar 4.22. Implementasi Metode Deteksi RCE pada *Challenge Dataset*

Pengujian kinerja ini dapat memberikan gambaran bagaimana kemampuan metode deteksi dalam mendeteksi serangan-serangan RCE, sebelum benar-benar diimplementasikan pada simulasi serangan atau bahkan di dunia nyata. Rekam jejak pengujian peforma metode deteksi menggunakan *dataset challenge* RCE dapat dilihat pada tabel di berikut ini.

Tabel 4.38. Pengujian Performa Metode Deteksi *Dataset Challenge* RCE

ID	Classifier Name	Chal. Total	Chal. Valid	Chal. Invalid	Accuracy	Chal. ToP	Chal. Top Sec.	Chal. Top Per Item	Margin	Corpus Rules
1	Support Vector Machine	8493	8078	415	95,11%	0:00:15	15	0,00177	0,3	TFF
2	Naive Bayes	8493	8078	415	95,11%	0:00:03	3	0,00035	0,3	TFF
3	Logistic Regression	8493	8078	415	95,11%	0:00:05	5	0,00059	0,3	TFF
4	Gradient Boosting	8493	8078	415	95,11%	0:00:11	11	0,0013	0,3	TFF
5	K-Nearest Neigbors	8493	8078	415	95,11%	0:01:30	90	0,0106	0,3	TFF
6	Gradient Boosting	8493	8181	312	96,33%	0:00:06	6	0,00071	0,5	FFF
7	Support Vector Machine	8493	8181	312	96,33%	0:00:13	13	0,00153	0,5	FFF
8	K-Nearest Neigbors	8493	8181	312	96,33%	0:01:55	115	0,01354	0,5	FFF
9	Naive Bayes	8493	8078	415	95,11%	0:00:02	2	0,00024	0,5	FFF
10	Logistic Regression	8493	8078	415	95,11%	0:00:04	4	0,00047	0,5	FFF
11	Naive Bayes	8493	3404	5089	40,08%	0:00:02	2	0,00024	0,5	TTT
12	Logistic Regression	8493	3404	5089	40,08%	0:00:03	3	0,00035	0,5	TTT
13	Gradient Boosting	8493	3404	5089	40,08%	0:00:05	5	0,00059	0,5	TTT
14	Support Vector Machine	8493	3404	5089	40,08%	0:00:12	12	0,00141	0,5	TTT
15	K-Nearest Neigbors	8493	3404	5089	40,08%	0:01:10	70	0,00824	0,5	TTT
16	Naive Bayes	8493	1223	7270	14,40%	0:00:04	4	0,00047	0,5	FTT
17	Logistic Regression	8493	1223	7270	14,40%	0:00:04	4	0,00047	0,5	FTT
18	Gradient Boosting	8493	1223	7270	14,40%	0:00:06	6	0,00071	0,5	FTT

ID	Classifier Name	Chal. Total	Chal. Valid	Chal. Invalid	Accuracy	Chal. ToP	Chal. Top Sec.	Chal. Top Per Item	Margin	Corpus Rules
19	Support Vector Machine	8493	1223	7270	14,40%	0:00:23	23	0,00271	0,5	FTT
20	K-Nearest Neigbors	8493	1223	7270	14,40%	0:01:59	119	0,01401	0,5	FTT
21	Support Vector Machine	8493	8181	312	96,33%	0:00:18	18	0,00212	1	FFF
22	K-Nearest Neigbors	8493	8168	325	96,17%	0:03:40	220	0,0259	1	FFF
23	Gradient Boosting	8493	8205	288	96,61%	0:00:05	5	0,00059	1	FFF
24	Naive Bayes	8493	8095	398	95,31%	0:00:02	2	0,00024	1	FFF
25	Logistic Regression	8493	8064	429	94,95%	0:00:03	3	0,00035	1	FFF
26	K-Nearest Neigbors	8493	3852	4641	45,35%	0:02:16	136	0,01601	1	TTT
27	Support Vector Machine	8493	3847	4646	45,30%	0:00:11	11	0,0013	1	TTT
28	Naive Bayes	8493	3434	5059	40,43%	0:00:02	2	0,00024	1	TTT
29	Logistic Regression	8493	3429	5064	40,37%	0:00:03	3	0,00035	1	TTT
30	Gradient Boosting	8493	3103	5390	36,54%	0:00:05	5	0,00059	1	TTT
31	Support Vector Machine	8493	8210	283	96,67%	0:00:22	22	0,00259	2	FFF
32	Gradient Boosting	8493	8230	263	96,90%	0:00:05	5	0,00059	2	FFF
33	Logistic Regression	8493	8107	386	95,46%	0:00:03	3	0,00035	2	FFF
34	Naive Bayes	8493	7476	1017	88,03%	0:00:02	2	0,00024	2	FFF
35	Support Vector Machine	8493	1229	7264	14,47%	0:00:21	21	0,00247	2	TTT
36	K-Nearest Neigbors	8493	1229	7264	14,47%	0:04:10	250	0,02944	2	TTT
37	Logistic Regression	8493	1628	6865	19,17%	0:00:03	3	0,00035	2	TTT

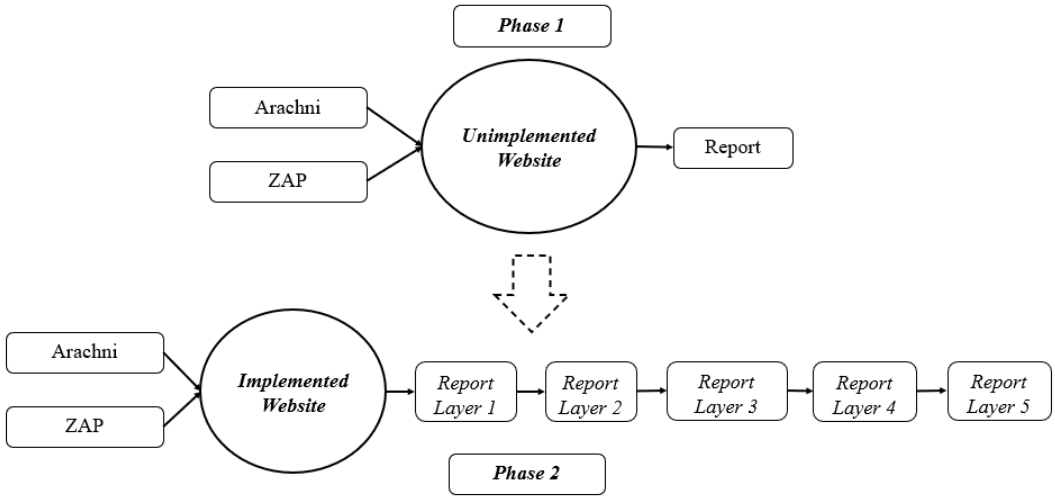
ID	Classifier Name	Chal. Total	Chal. Valid	Chal. Invalid	Accuracy	Chal. ToP	Chal. Top Sec.	Chal. Top Per Item	Margin	Corpus Rules
38	Gradient Boosting	8493	3392	5101	39,94%	0:00:05	5	0,00059	2	TTT
39	K-Nearest Neigbors	8493	1020	7473	12,01%	0:04:55	295	0,03473	2	FFF
40	Naive Bayes	8493	1273	7220	14,99%	0:00:02	2	0,00024	2	TTT
41	Support Vector Machine	8493	8238	255	97,00%	0:00:17	17	0,002	3	FFF
42	Gradient Boosting	8493	8269	224	97,36%	0:00:05	5	0,00059	3	FFF
43	Logistic Regression	8493	8155	338	96,02%	0:00:03	3	0,00035	3	FFF
44	K-Nearest Neigbors	8493	1043	7450	12,28%	0:07:28	448	0,05275	3	FFF
45	Naive Bayes	8493	7518	975	88,52%	0:00:03	3	0,00035	3	FFF
46	Support Vector Machine	8493	8240	253	97,02%	0:00:20	20	0,00235	4	FFF
47	K-Nearest Neigbors	8493	8237	256	96,99%	0:13:24	804	0,09467	4	FFF
48	Logistic Regression	8493	7529	964	88,65%	0:00:04	4	0,00047	4	FFF
49	Gradient Boosting	8493	8187	306	96,40%	0:00:05	5	0,00059	4	FFF
50	Naive Bayes	8493	7518	975	88,52%	0:00:04	4	0,00047	4	FFF
51	K-Nearest Neigbors	8493	3385	5108	39,86%	0:15:12	912	0,10738	5	FFF
52	Support Vector Machine	8493	3558	4935	41,89%	0:00:19	19	0,00224	5	FFF
53	Gradient Boosting	8493	3503	4990	41,25%	0:00:06	6	0,00071	5	FFF
54	Logistic Regression	8493	2849	5644	33,55%	0:00:04	4	0,00047	5	FFF
55	Naive Bayes	8493	711	7782	8,37%	0:00:04	4	0,00047	5	FFF
56	K-Nearest Neigbors	8493	1240	7253	14,60%	7:39:00	27540	3,24267	5	TTT

ID	Classifier Name	Chal. Total	Chal. Valid	Chal. Invalid	Accuracy	Chal. ToP	Chal. Top Sec.	Chal. Top Per Item	Margin	Corpus Rules
57	Support Vector Machine	8493	1241	7252	14,61%	0:00:35	35	0,00412	5	TTT
58	Logistic Regression	8493	285	8208	3,36%	0:00:04	4	0,00047	5	TTT
59	Gradient Boosting	8493	3379	5114	39,79%	0:00:05	5	0,00059	5	TTT
60	Naive Bayes	8493	1197	7296	14,09%	0:00:04	4	0,00047	5	TTT
61	Support Vector Machine	8493	3944	4549	46,44%	0:00:15	15	0,00177	6	TFF
62	K-Nearest Neigbors	8493	3719	4774	43,79%	0:17:10	1030	0,12128	6	TFF
63	Logistic Regression	8493	3332	5161	39,23%	0:00:04	4	0,00047	6	TFF
64	Gradient Boosting	8493	4886	3607	57,53%	0:00:05	5	0,00059	6	TFF
65	Naive Bayes	8493	2323	6170	27,35%	0:00:05	5	0,00059	6	TFF
66	Support Vector Machine	8493	5670	2823	66,76%	0:00:21	21	0,00247	7	TFF
67	K-Nearest Neigbors	8493	5167	3326	60,84%	0:25:41	1541	0,18144	7	TFF
68	Logistic Regression	8493	3335	5158	39,27%	0:00:06	6	0,00071	7	TFF
69	Gradient Boosting	8493	4881	3612	57,47%	0:00:09	9	0,00106	7	TFF
70	Naive Bayes	8493	2321	6172	27,33%	0:00:09	9	0,00106	7	TFF
71	Support Vector Machine	8493	6096	2397	71,78%	0:00:29	29	0,00341	8	TFF
72	K-Nearest Neigbors	8493	3454	5039	40,67%	0:29:16	1756	0,20676	8	TFF
73	Logistic Regression	8493	3058	5435	36,01%	0:00:05	5	0,00059	8	TFF
74	Gradient Boosting	8493	4887	3606	57,54%	0:00:06	6	0,00071	8	TFF
75	Naive Bayes	8493	2361	6132	27,80%	0:00:06	6	0,00071	8	TFF

ID	Classifier Name	Chal. Total	Chal. Valid	Chal. Invalid	Accuracy	Chal. ToP	Chal. Top Sec.	Chal. Top Per Item	Margin	Corpus Rules
76	Support Vector Machine	8493	6747	1746	79,44%	0:00:32	32	0,00377	9	TFF
77	Logistic Regression	8493	3049	5444	35,90%	0:00:05	5	0,00059	9	TFF
78	K-Nearest Neigbors	8493	2435	6058	28,67%	0:53:07	3187	0,37525	9	TFF
79	Gradient Boosting	8493	3564	4929	41,96%	0:00:06	6	0,00071	9	TFF
80	Naive Bayes	8493	2261	6232	26,62%	0:00:07	7	0,00082	9	TFF
81	Support Vector Machine	8493	7365	1128	86,72%	0:00:39	39	0,00459	10	TFF
82	K-Nearest Neigbors	8493	3810	4683	44,86%	0:35:24	2124	0,25009	10	TFF
83	Logistic Regression	8493	2564	5929	30,19%	0:00:06	6	0,00071	10	TFF
84	Gradient Boosting	8493	3102	5391	36,52%	0:00:07	7	0,00082	10	TFF
85	Naive Bayes	8493	1526	6967	17,97%	0:00:08	8	0,00094	10	TFF
86	**Support Vector Machine**	**8493**	**8480**	**13**	**99,85%**	**0:00:37**	**37**	**0,00436**	**15**	**TFF**
87	K-Nearest Neigbors	8493	3975	4518	46,80%	0:53:52	3232	0,38055	15	TFF
88	Logistic Regression	8493	7423	1070	87,40%	0:00:06	6	0,00071	15	TFF
89	Gradient Boosting	8493	5071	3422	59,71%	0:00:07	7	0,00082	15	TFF
90	Naive Bayes	8493	5532	2961	65,14%	0:00:11	11	0,0013	15	TFF

4.2.2. Implementasi Metode Mitigasi *Multi-Layer Security*

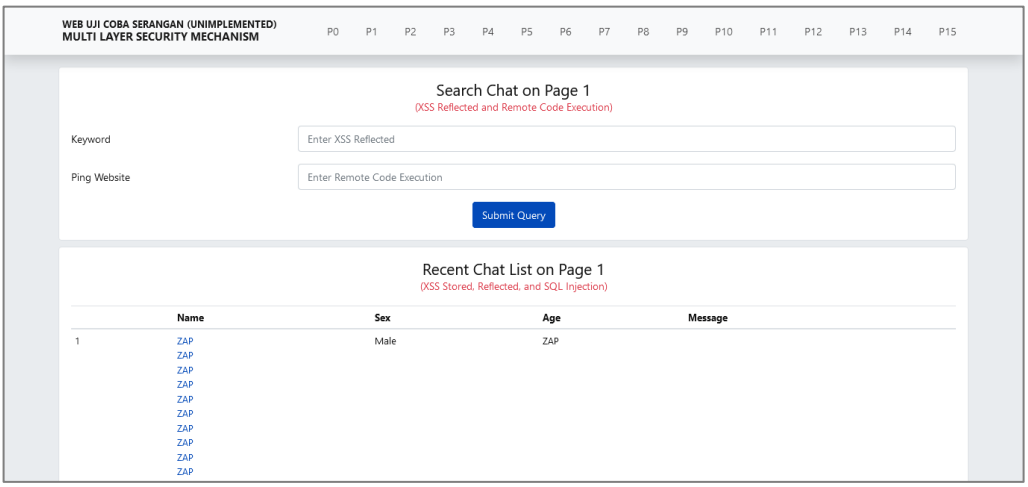
Metode mitigasi *multi-layer security* telah diujikan pada *website* uji coba. Dalam konteks *security testing*, pengujian metode mitigasi ini menggunakan metode *white box testing*. Pada metode ini, celah keamanan atau *web vulnerabilities* telah sengaja disediakan, sehingga tingkat efektivitas metode mitigasi dapat terukur secara tegas. Celah keamanan yang disediakan adalah XSS, SQLi, dan RCE. Konsep web uji coba ini serupa dengan DVWA (*Damn Vulnerable Web Application*) yang biasa digunakan untuk melakukan *security testing*.



Gambar 4.23. Fase-fase Simulasi Serangan pada Web Ujicoba

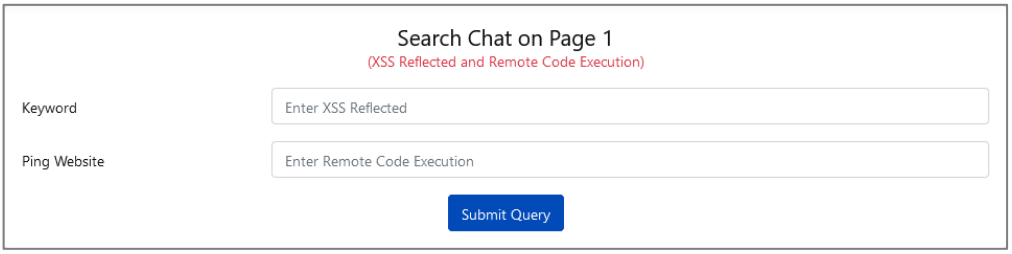
Web uji coba dibangun dengan Python Django. Django dipilih karena *web framework* ini telah menyediakan *built-in function* yang kompatibel dengan metode *multi-layer security* yang diusulkan. Skenario umum pengujian metode mitigasi terdiri dari dua tahapan, yaitu (1) melakukan simulasi penyerangan pada tahapan *unimplemented*, yaitu tahapan ketika web uji coba belum menerapkan metode mitigasi dan (2) melakukan simulasi penyerangan pada setiap lapisan metode mitigasi dan mengukur atau mengkalkulasi tingkat efektivitasnya (gambar 4.23).

Baik Arachni maupun ZAP melakukan metode *web crawling* untuk mencari celah keamanan web. Kedua aplikasi ini mengakses seluruh URL yang ditemukan di *website target* secara *recursive*. Setiap URL yang di-*crawling* diuji dengan vektor-vektor serangan yang tersimpan di database masing-masing aplikasi. Oleh karena itu, *website* uji coba menyediakan URL yang *vulnerable*, sehingga kedua aplikasi penyerang dapat melakukan penyerangan secara komprehensif. Berikut ini adalah tampilan website uji coba yang digunakan dalam penelitian.

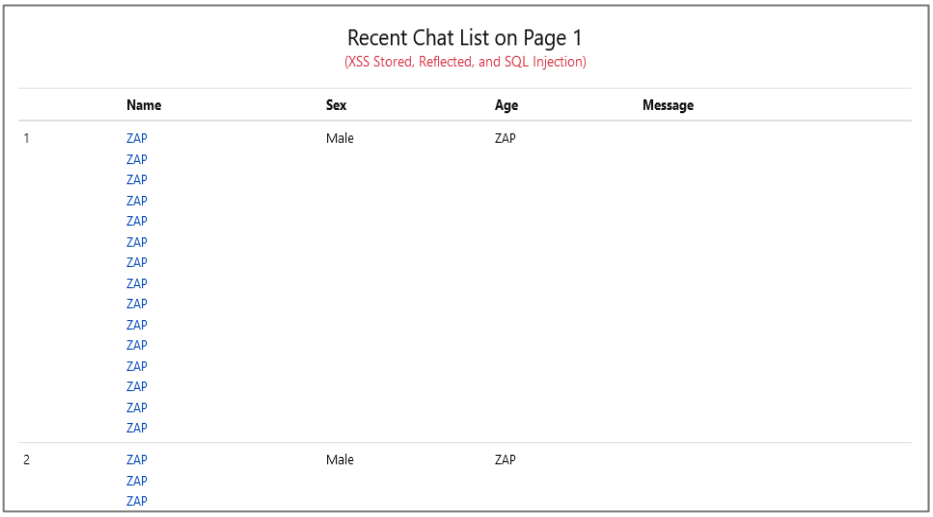


Gambar 4.24. Tampilan Website Ujicoba dengan Konsep Aplikasi *Chat*

Seperti yang tertera pada gambar tersebut, pada navigasi atas, terdapat menu dengan nama P0, P1, P2, P3, …., P15. Huruf P di dalam navigasi tersebut bermakna *page* atau laman, yang berarti terdapat 16 laman di *website* uji coba tersebut. Pada tiap laman tersebut, terdapat elemen *website* yang *vulnerable*, sehingga dapat digunakan oleh aplikasi penyerang untuk mengirimkan kode serangan.



Gambar 4.25. Formulir untuk Mengirimkan Serangan *Reflected XSS* dan RCE



Gambar 4.26. *View* untuk Mempermudah Aplikasi Memverifikasi Serangan

Gambar 4.72 di atas merupakan elemen *view* yang dapat memermudah aplikasi penyerang dalam memverifikasi status penyerangan, apakah sukses atau gagal dalam mengeksekusi suatu serangan. Elemen *website* ini disediakan untuk serangan *Stored XSS, Refleced XSS,* dan SQLi. Pembahasan impelementasi masing-masing teknik serangan dapat dilihat pada pemaparan 4.2.2.1, 4.2.2.2, dan 4.2.2.3. Penelitian ini mengimplementasikan serangan *multi-website* dengan distribusi berikut.

Tabel 4.39. Distribusi *Multi-Website* Berdasarkan Celah Keamanan

No	Website ID	Celah Keamanan
1	XSS	Cross Site Scripting
2	SQLi	SQL Injection
3	RCE	Remote Code Execution

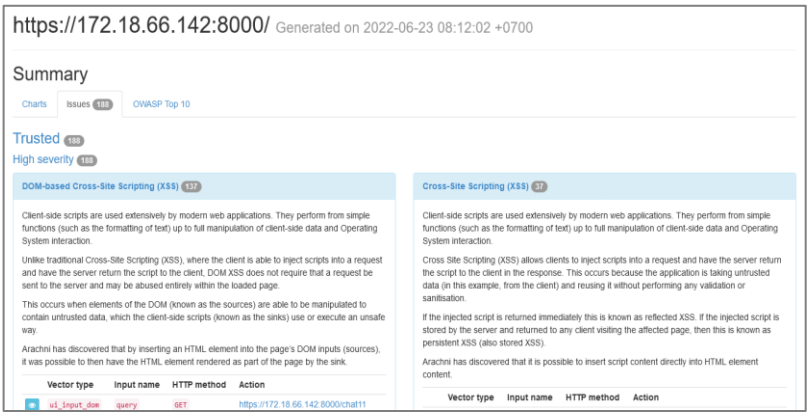
4.2.2.1. Teknik Serangan *Cross Site Scripting* (XSS)

Untuk mengukur tingkat efektivitas metode mitigasi dalam memitigasi serangan XSS, sebanyak 336 celah keamanan XSS disediakan pada website uji coba. Rincian celah keamanan XSS tersebut adalah 240 DOM XSS, 48 *Stored XSS*, dan 48 *Reflected XSS*. Untuk diketahui bahwa Arachni dan ZAP menghitung celah kea-

manan berdasarkan keunikan laman, bukan berdasarkan vektor serangan yang digunakan. Artinya, meskipun terdapat lebih dari satu vektor serangan yang dapat dilakukan di laman *website*, kedua aplikasi tersebut tetap menghitungnya sebagai satu celah keamanan. Dengan kata lain, meskipun terdapat celah keamanan yang disediakan di *website* uji coba, jumlah vektor serangan yang dapat dilakukan bisa jadi lebih dari 336 vektor serangan. Pada kasus serangan XSS misalnya, satu celah keamanan yang XSS dapat dieksploitasi dengan banyak vektor serangan. Berikut ini adalah hasil implementasi metode mitigasi serangan XSS.

1) Tahapan *Unimplemented*

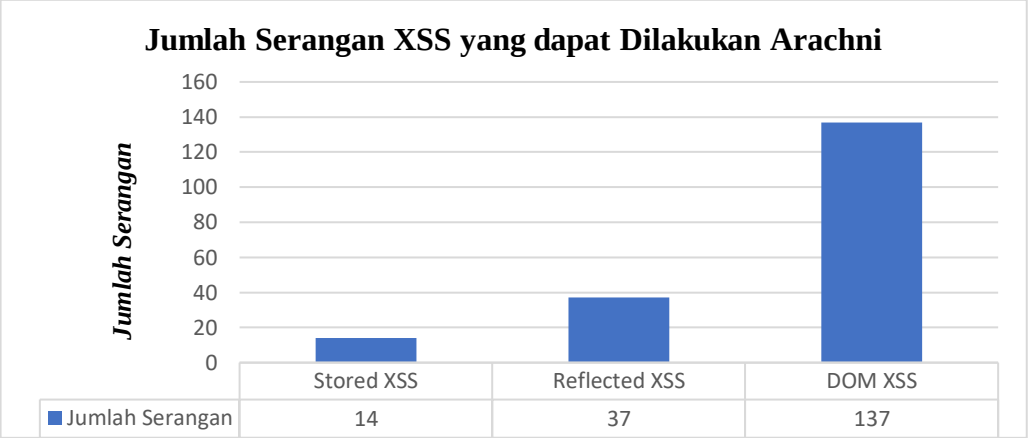
Pada saat metode mitigasi belum diimplementasikan, terdapat 188 serangan XSS yang dapat dilakukan Arachni. Rincian jenis serangan XSS yang dapat dilakukan adalah 137 DOM *XSS*, 37 *Reflected XSS*, dan 14 *Stored XSS*. Sementara itu, ZAP hanya mampu melakukan 38 *Stored XSS*. Dengan demikian, Arachni mampu melakukan serangan XSS lebih banyak dari ZAP. Berikut ini adalah tangkapan layar tampilan laporan Arachni dan ZAP terkait serangan XSS.



Gambar 4.27. Tangkapan Layar Laporan Serangan XSS Arachni

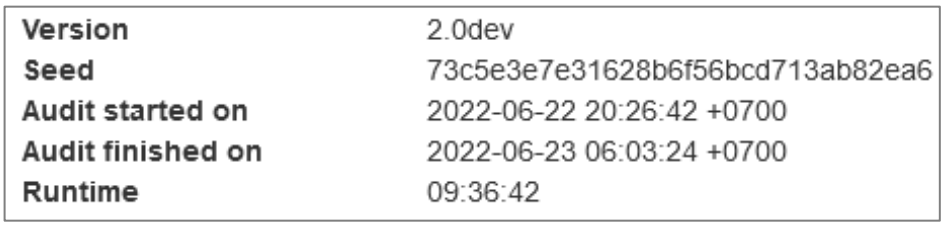
Arachni melakukan klasifikasi pada setiap serangan yang berhasil dilakukan, berdasarkan pada tingkat risikonya, mulai dari *low*, *medium*, dan *high-severity*. Se-

rangan XSS sendiri berada pada klasifikasi serangan *high-severity*. Artinya, serangan XSS ini termasuk dalam jenis serangan dengan tingkat kerusakan tinggi dan berbahaya jika tidak dimitigasi. Laporan Arachni dilengkapi dengan rincian URL yang *vunerable*. Berikut ini adalah rekapitulasi visual serangan XSS oleh Arachni.

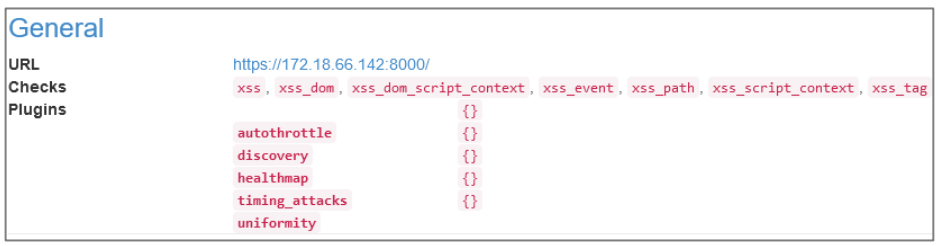


Grafik 4.48. Jumlah Serangan XXS yang Dilakukan Arahcni

Ketika melakukan serangan, Arachni menggunakan profil XSS dengan waktu eksekusi selama 09:36:42. Serangan XSS Arachni dilakukan secara mendalam. Untuk menyelesaikan serangan, Arachni memerlukan durasi waktu serangan lebih dari sembilan jam. Parameter konfigurasi umum Arachni dan lama waktu *runtime* Arachni saat melakukan serangan dapat dilihat pada gambar di bawah.



Gambar 4.28. Waktu *Runtime* Arachni dalam Melakukan Serangan

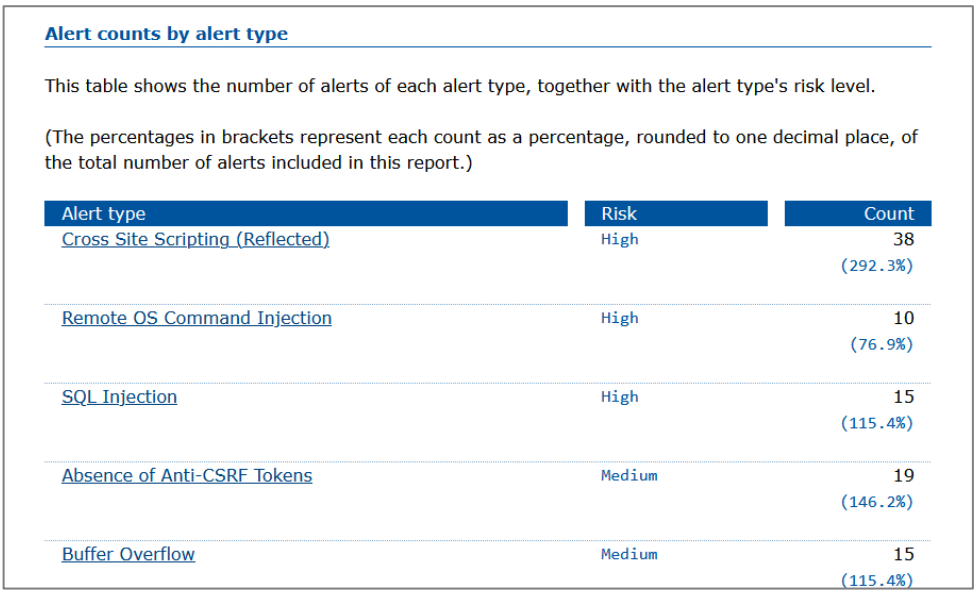


Gambar 4.29. Parameter Konfigurasi Arachni Ketika Melakukan Serangan XSS

Berbeda dengan Arachni, ZAP hanya mampu menghasilkan 38 serangan *Reflected XSS*. Saat melakukan serangan, mode *security testing* yang digunakan Arachni adalah *Attack Mode* atau mode serangan. Selain menggunakan mode serangan, proses *web crawling* juga menggunakan *headless browser*. Dalam hal ini, *headless browser* yang digunakan adalah Modzila Firefox.

Gambar 4.30. Konfigurasi ZAP Ketika Melakukan Serangan

Setelah melakukan serangan, pengguna ZAP dapat melakukan *generate report* untuk melihat hasil serangan, mulai dari URL yang *vulnerable*, jenis celah keamanan yang terbuka, dan lain-lain. Berikut ini adalah rekapitulasi hasil serangan yang dilakukan oleh ZAP. Seperti yang tertera pada gambar, serangan XSS termasuk dalam serangan kategori berrisiko tinggi.



Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
Cross Site Scripting (Reflected)	High	38 (292.3%)
Remote OS Command Injection	High	10 (76.9%)
SQL Injection	High	15 (115.4%)
Absence of Anti-CSRF Tokens	Medium	19 (146.2%)
Buffer Overflow	Medium	15 (115.4%)

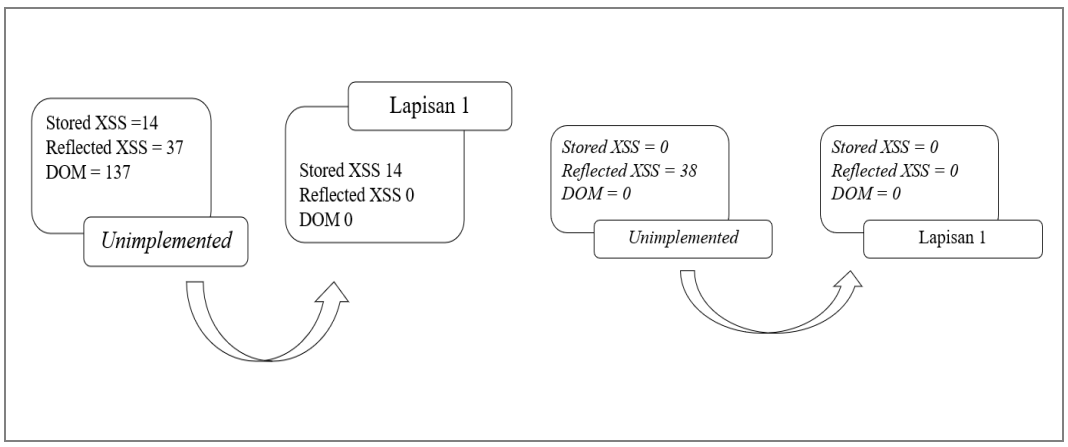
Gambar 4.31. Rekapitulasi Hasil Serangan ZAP ke Website Ujicoba

2) Tahapan *Implemented*

Setelah melakukan simulasi serangan pada tahapan *unimplemented*, serangan dilakukan pada tahapan *implemented*. Pada tahapan ini, metode mitigasi diimplementasikan untuk mengukur tingkat efektivitas dalam memitigasi serangan XSS. Simulasi serangan ke *web target* dilakukan secara bertahap, mulai dari lapisan pertama sampai dengan kelima atau berhenti pada lapisan yang mampu memitigasi seluruh serangan tipe XSS. Artinya, saat lapisan di dalam metode mitigasi sukses memitigasi serangan, pengukuran tingkat efektivitas berhenti sampai lapisan itu.

a) Lapisan Pertama (OWASP *ModSecurity)*

Setelah lapisan pertama diimplementasikan, serangan XSS kembali dilakukan. Pada tahapan ini, OWASP ModSecurity mengaktifkan *core rule set* XSS yang bernama REQUEST-941-APPLICATION-ATTACK-XSS.conf. Rincian hasil serangan setelah lapisan pertama diimplementasikan adalah sebagai berikut.



Gambar 4.32. Hasil Implementasi Mitigasi Serangan XSS pada Lapisan Pertama

Setelah lapisan pertama diimplementasikan, serangan DOM XSS dan *Reflected* XSS tidak dapat dilakukan. Sebelum diimplementasikan, jumlah serangan *Reflected* XSS adalah 14 dan DOM XSS 137 serangan. Namun, lapisan pertama gagal

dalam memitigasi serangan *Stored XSS*. Karena masih ada serangan yang belum termitigasi, lapisan kedua kemudian diimplementasikan. *Sample rule* lapisan pertama untuk memitigasi serangan XSS adalah berikut.

```
#
# -= Paranoia Level 0 (empty) =- (apply unconditionally)
#

SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1" "id:941011,phase:1,pass,nolog,skipAfter:END-REQUEST-941-APPLICATION-ATTACK-XSS"
SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1" "id:941012,phase:2,pass,nolog,skipAfter:END-REQUEST-941-APPLICATION-ATTACK-XSS"
#
# -= Paranoia Level 1 (default) =- (apply only when tx.detection_paranoia_level is sufficiently high: 1 or higher)
#



#
# -=[ Libinjection - XSS Detection ]=-
#
# Ref: https://github.com/client9/libinjection
# Ref: https://speakerdeck.com/ngalbreath/libinjection-from-sqli-to-xss
#
# -=[ Targets ]=-
#
# 941100: PL1 : REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|
#               REQUEST_COOKIES_NAMES|REQUEST_HEADERS:User-Agent|
#               ARGS_NAMES|ARGS|XML:/*
#
# 941101: PL2 : REQUEST_FILENAME|REQUEST_HEADERS:Referer
#
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|REQUEST_HEADERS:User-Agent|ARGS_NAMES|ARGS|XML:/* "@detectXSS" \
    "id:941100,\
    phase:2,\
    block,\

t:none,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,t:removeNulls,\
    msg:'XSS Attack Detected via libinjection',\
    logdata:'Matched Data: XSS data found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}',\
    tag:'application-multi',\
    tag:'language-multi',\
    tag:'platform-multi',\
    tag:'attack-xss',\
    tag:'paranoia-level/1',\
    tag:'OWASP_CRS',\
    tag:'capec/1000/152/242',\
    ctl:auditLogParts=+E,\
    ver:'OWASP_CRS/4.0.0-rc1',\
    severity:'CRITICAL',\
    setvar:'tx.xss_score=+%{tx.critical_anomaly_score}',\
    setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
```

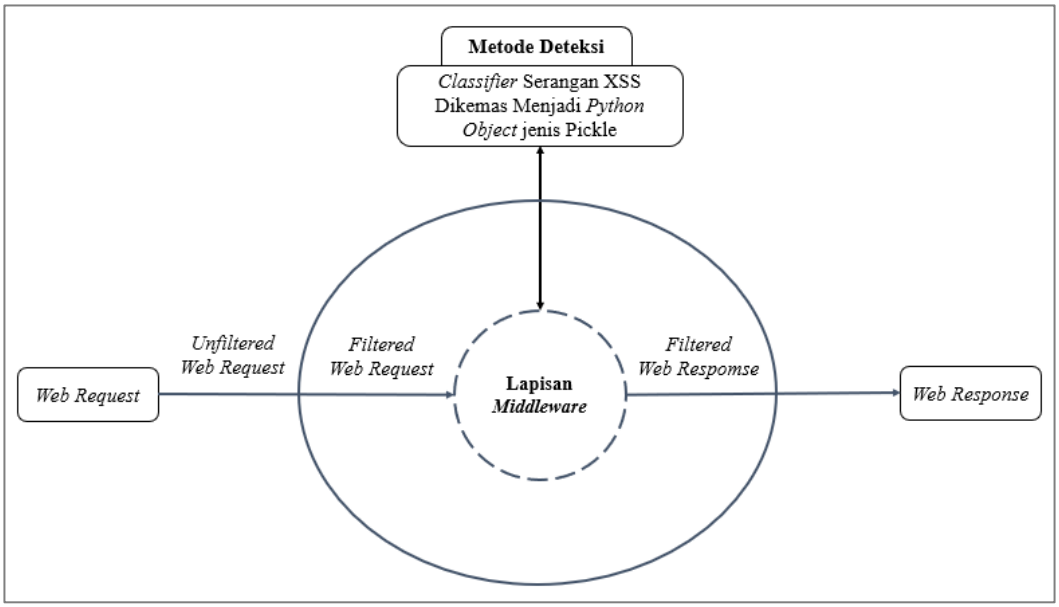
```
45.
46.
47. #
48. # -=[ XSS Filters - Category 1 ]=-
49. # http://xssplayground.net23.net/xssfilter.html
50. # script tag based XSS vectors, e.g., <script> alert(1)</script>
51. #
52. SecRule
    REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|REQUEST_FILENA
    ME|REQUEST_HEADERS:User-Agent|REQUEST_HEADERS:Referer|ARGS_NAMES|ARGS|XML:/*
    "@rx (?i)<script[^>]*>[\s\S]*?" \
53.     "id:941110,\
54.     phase:2,\
55.     block,\
56.     capture,\
57.
    t:none,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDeco
    de,t:removeNulls,\
58.     msg:'XSS Filter - Category 1: Script Tag Vector',\
59.     logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}:
    %{MATCHED_VAR}',\
60.     tag:'application-multi',\
61.     tag:'language-multi',\
62.     tag:'platform-multi',\
63.     tag:'attack-xss',\
64.     tag:'paranoia-level/1',\
65.     tag:'OWASP_CRS',\
66.     tag:'capec/1000/152/242',\
67.     ctl:auditLogParts=+E,\
68.     ver:'OWASP_CRS/4.0.0-rc1',\
69.     severity:'CRITICAL',\
70.     setvar:'tx.xss_score=+%{tx.critical_anomaly_score}',\
71.     setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
72.
73.
74. #
75. # -=[ XSS Filters - Category 3 ]=-
76. #
77. # Regular expression generated from util/regexp-assemble/data/941130.data.
78. # To update the regular expression run the following shell script
79. # (consult util/regexp-assemble/README.md for details):
80. #   util/regexp-assemble/regexp-assemble.py update 941130
81. #
82. SecRule
    REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|REQUEST_HEADER
    S:User-Agent|ARGS_NAMES|ARGS|XML:/* "@rx
    (?i)[\s\S](?:\b(?:x(?:link:href|html|mlns)|data:text\/html|pattern\b.*?=|form
    action)|!ENTITY\s+(?:\S+|%\s+\S+)\s+(?:PUBLIC|SYSTEM)|;base64|@import)\b" \
83.     "id:941130,\
84.     phase:2,\
85.     block,\
86.     capture,\
87.
    t:none,t:utf8toUnicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDeco
    de,t:removeNulls,\
88.     msg:'XSS Filter - Category 3: Attribute Vector',\
89.     logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}:
    %{MATCHED_VAR}',\
90.     tag:'application-multi',\
91.     tag:'language-multi',\
92.     tag:'platform-multi',\
93.     tag:'attack-xss',\
94.     tag:'paranoia-level/1',\
95.     tag:'OWASP_CRS',\
```

```
96.     tag:'capec/1000/152/242',\
97.     ctl:auditLogParts=+E,\
98.     ver:'OWASP_CRS/4.0.0-rc1',\
99.     severity:'CRITICAL',\
100.      setvar:'tx.xss_score=+%{tx.critical_anomaly_score}',\
101.      setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
```

Gambar 4.33. *Core Rule Set* Serangan XSS

b) Lapisan Kedua (*HTTP Middleware*)

Lapisan kedua diimplementasikan untuk memitigasi serangan *Stored* XSS. Pada lapisan ini, Django *Middleware* digunakan untuk memitigasi serangan XSS. *Request* yang diterima dan *response* yang dikirimkan diproses terlebih dahulu pada lapisan *middleware*. Oleh karena itu, metode deteksi yang dihasilkan telah diletakan pada lapisan ini. Terdapat tiga hal yang dilakukan peneliti untuk memitigasi serangan XSS pada lapisan *middleware*, yaitu sebagai berikut.



Gambar 4.34. Implementasi Metode Deteksi pada Lapisan *Middleware*

Pertama, metode deteksi yang telah melalui proses *training* dan *testing* dikemas dalam bentuk Python *Object* atau Pickle sehingga dapat dieksekusi lebih fleksibel. Kedua, metode deteksi diletakan di lapisan *middleware*. Ketiga, semua *request* yang masuk diproses oleh metode deteksi. Jika *request* mengandung kode sera-

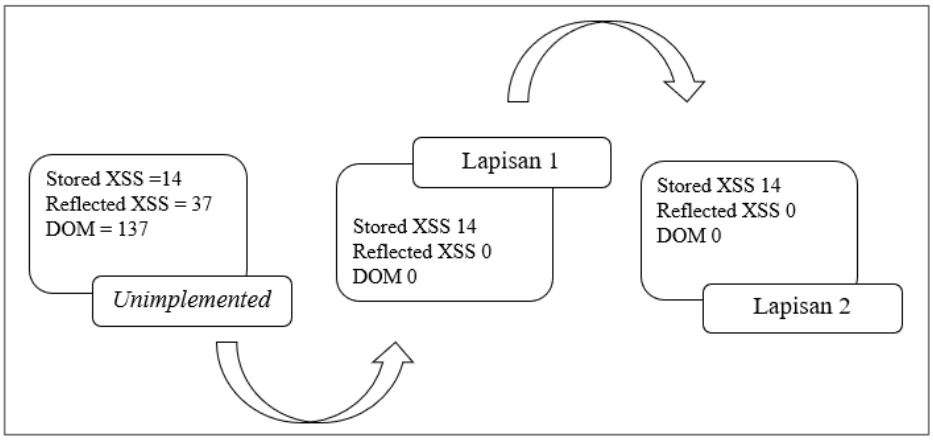
ngan XSS, maka aplikasi akan mengirimkan status kode HTTP 404 atau menandakan bahwa laman yang di-*request* tidak ditemukan. Pengiriman status kode 404 untuk mengisoliasi laman *website* dari upaya serangan. Selain itu, status kode HTTP 404 juga dapat mengelabui penyerang, karena penyerang tidak dapat memvalidasi keberhasilan penyerangan. Status ini membuat seolah-olah laman tak tersedia. Berikut ini adalah implementasi metode deteksi di lapisan *middleware*.

```
38. import os
39. from django.http import Http404
40. from django.conf import settings
41. from aurora.backend.library.mlsm.xss.classifier import XSSClassifier
42. from aurora.backend.library.mlsm.rce.classifier import RCEClassifier
43. from aurora.backend.library.mlsm.sqli.classifier import SQLiClassifier
44.
45.
46. class MSLMMiddleware(object):
47.
48.     def __init__(self, get_response):
49.         self.get_response = get_response
50.
51.
52.     def __call__(self, request):
53.         response = self.get_response(request)
54.         return self.process_request(request, response)
55.
56.
57.     def process_request(self, request, response):
58.         GET_DATA  = request.get_full_path()
59.         POST_DATA = request.POST
60.         if settings.LAYER_2_MIDLEWARE == True:
61.             # Cross Site Scripting Detection
62.             # Remote Code Execution Detection
63.             # SQL Injection Detection
64.             classifiers = {
65.                 'xss'   : XSSClassifier(),
66.                 'rce'   : RCEClassifier(),
67.                 'sqli'  : SQLiClassifier()
68.             }
69.             for data in [GET_DATA, POST_DATA]:
70.                 for name, classifier in classifiers.items():
71.                     label = classifier.classify(data)
72.                     if label == 'payload':
73.                         raise Http404()
74.         return response
```

Gambar 4.35. Kode Python Mengaktifkan Metode Deteksi Lapisan *Middleware*

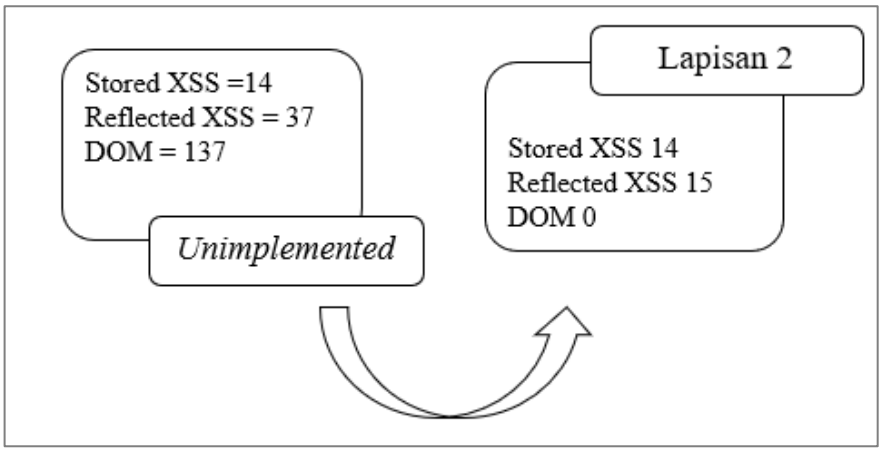
Seperti yang terlihat pada kode Python di atas, metode deteksi diletakan pada baris 28, disertai dengan *classifier* yang lainnya. Ketika data dari GET atau POST terdeteksi sebagai *payload*, maka aplikasi akan mengirimkan kode status HTTP 404

pada penyerang. Serangan XSS kembali dilakukan setelah lapisan kedua diaktifkan. Namun, berdasarkan penelitian yang telah dilakukan, lapisan ini belum mampu memitigasi serangan *Stored XSS*. Sebanyak 14 *Stored XSS* masih dapat dilakukan meskipun lapisan kedua ini telah diaktifkan.



Gambar 4.36. Hasil Implementasi Mitigasi XSS Lapisan Kedua (Integratif)

Ketidakberhasilan lapisan kedua yang terkait dengan metode deteksi menimbulkan skeptis penelitian, terutama terkait tingkat akurasi metode deteksi. Oleh karena itu, pengujian lapisan kedua juga dilakukan secara *standalone*. Artinya, lapisan kedua diaktifkan, namun lapisan pertama tidak diaktifkan. Hal ini dilakukan untuk mengukur efektivitas lapisan kedua secara independen, tanpa lapisan pertama. Berikut ini hasil implementasi serangan XSS pada lapisan kedua secara *standalone*.



Gambar 4.37. Hasil Implementasi Mitigasi XSS Lapisan Kedua (Standalone)

Berdasarkan uji coba yang dilakukan, lapisan kedua ternyata mampu memitigasi 137 serangan DOM XSS dan 22 serangan *Stored XSS*. Pada uji coba secara *standalone* tersebut, *Stored XSS* memang tidak dapat termitigasi. Oleh karena itu, karena masih ada serangan XSS yang belum dapat dimitigasi oleh lapisan pertama dan kedua, lapisan ketiga kemudian diimplementasikan, yaitu ***template engine*.**

c) Lapisan Ketiga (*Template Engine*)

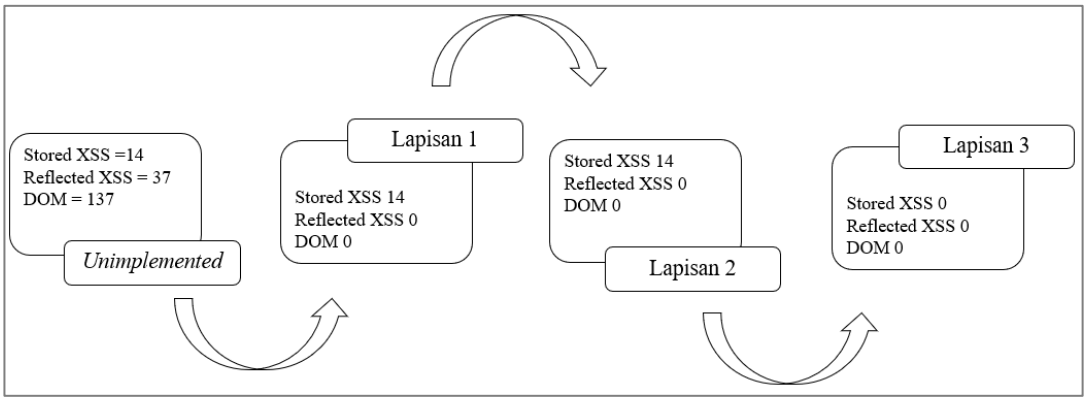
Untuk mengimplementasikan lapisan ketiga, peneliti menggunakan *template engine* yang merupakan *template engine* bawaan Django. Sebelum lapisan ini diimplementaiskan, setiap kode HTML/Javascript di berkas template menggunakan pengaturan *autoescape off*, untuk menguji efektivitasnya. Oleh karena itu, ketika lapisan ketiga diimplementasikan, *autoescape* diubah menjadi *on*. Berikut ini adalah kode *template* yang digunakan untuk memitigasi serangan *Stored XSS*.

```
1.  <div class='p-3 my-2 border bg-white rounded'>
2.      <h4 class="text-center m-0">Recent Chat List on Page {{ page }}</h4>
3.      <div class="text-danger mb-3 text-center">(XSS Stored, Reflected,
    and SQL Injection)</div>
4.      {% if keyword %}
5.          {% autoescape on %}
6.              <p>Chat result for keyword <b>{{ keyword }}</b></p>
7.          {% endautoescape %}
8.      {% endif %}
9.      {% if query %}
10.         {% autoescape on %}
11.             <p>You search for this file <b>{{ query }}</b></p>
12.         {% endautoescape %}
13.     {% endif %}
14.     {% if response %}
15.         {% autoescape on %}
16.             <p>You search for this response? <div class='rce-
    query'><b>{{ response }}</b></div></p>
17.         {% endautoescape %}
18.     {% endif %}
19. </div>
```

Gambar 4.38. Implementasi Mitigasi Serangan XSS pada Lapisan Ketiga

Implementasi *template engine* dapat dilihat pada setiap elemen HTML yang diapit dengan fungsi *autoescape*. Setiap kode HTML/Javascript yang diapit oleh

fungsi ini akan di-*escape*. Artinya, kode tersebut tidak akan dieksekusi oleh *browser* kecuali pengguna atau *web developer* memang menginginkannya. Dengan metode ini, kode Javascript tidak akan dieksekusi secara otomatis. Berikut ini adalah hasil setelah lapisan ketiga diiimplementasikan.



Gambar 4.39. Hasil Implementasi Mitigasi Serangan XSS pada Lapisan Ketiga

Berdasarkan uji coba serangan XSS yang telah dilakukan, lapisan ketiga berhasil dalam memitigasi serangan *Stored XSS*. Tidak ada satupun serangan *Stored* XSS yang dapat dilakukan. Temuan ini sekaligus menutup proses mitigasi serangan XSS yang cukup dilakukan pada lapisan ketiga. Suatu kesimpulan dapat ditarik dari temuan ini, yaitu tiga lapisan *multi-layer security* harus diaktifkan untuk memitigasi serangan XSS jenis *stored, reflected*, dan DOM.

4.2.2.2. Teknik Serangan *SQL Injection (SQLi)*

Sebanyak 16 celah keamanan SQLi disediakan di *website* uji coba untuk mengetahui tingkat efektivitas implementasi metode mitigasi dalam mengatasi serangan SQLi. Arachni dan ZAP memiliki *built-in functions* atau profil untuk melakukan serangan SQLi secara masif dan komprehensif. Serangan SQLi termasuk dalam serangan kategori *high-severity* atau tingkat kerusakan tinggi, terutama jika penyerang berhasil mengakses database *users* atau pengguna yang menghimpun seluruh

akun pengguna *website*. Selain itu, celah keamanan ini juga memungkinkan terjadinya *data breach* atau akses data secara ilegal. Jika *website* menghimpun data-data rahasia dengan tingkat kerahasiaan yang tinggi, maka data-data tersebut berpotensi untuk dicuri dan disalahgunakan oleh pihak-pihak yang tak bertanggung jawab. Berikut ini adalah hasil implementasi metode mitigasi serangan SQLi.

1) Tahapan *Unimplemented*

Ketika metode mitigasi belum diimplementasikan, terdapat 15 serangan SQLi yang dapat dilakukan oleh Arachni. Sama halnya dengan Arachni, ZAP juga menemukan 15 celah keamanan SQLi pada *website* uji coba. Seperti yang telah disampaikan pada 4.2.2.1, satu celah keamanan dapat menjadi *hole* untuk berbagai macam vektor serangan. Oleh karena itu, 15 celah keamanan SQLi dapat beresiko tinggi pada keamanan dan kestabilan *website*. Berikut ini adalah tampilan Arachni dan ZAP ketika melakukan serangan SQLi pada tahap *unimplemented*.

Gambar 4.40. Tangkapan Layar Laporan Serangan SQLi Arachni

Seperti yang tertera pada gambar tersebut, SQLi termasuk pada serangan dengan kategori tingkat kerusakan tinggi. Sementara itu, hasil rekapitulasi serangan SQLi yang dilakukan ZAP adalah sebagai berikut.

Alert type	Risk	Count
Cross Site Scripting (Reflected)	High	38 (292.3%)
Remote OS Command Injection	High	10 (76.9%)
SQL Injection	High	15 (115.4%)
Absence of Anti-CSRF Tokens	Medium	19 (146.2%)
Buffer Overflow	Medium	15 (115.4%)

Gambar 4.41. Tangkapan Layar Laporan Serangan SQLi ZAP

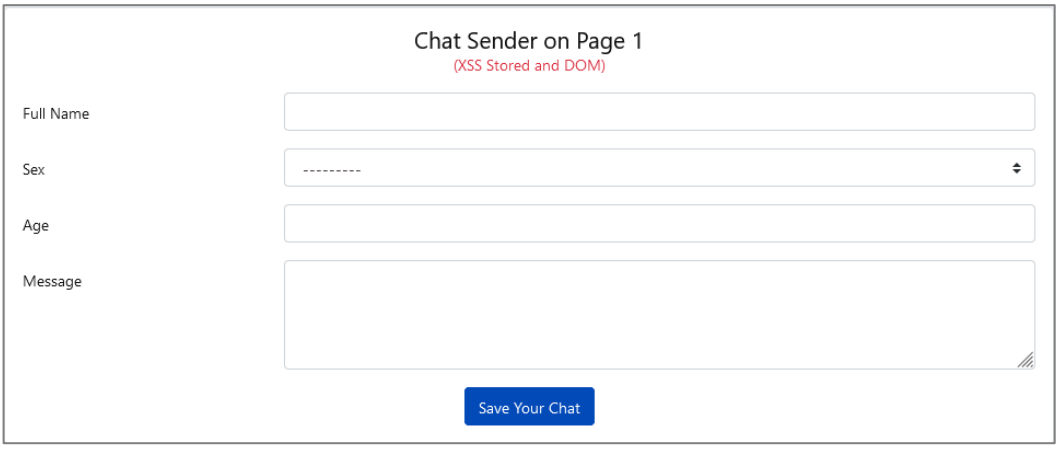
Arachni dan ZAP menggunakan metode *web crawling* untuk mencari celah keamanan SQLi. Berikut ini adalah daftar URL yang terdeteksi *vulnerable* terhadap serangan SQLI yang ditemukan oleh Arachni dan ZAP.

Tabel 4.40. Daftar URL yang *Vulnerable* terhadap Serangan SQLi

No	*Vector type*	*Input name*	*HTTP method*	*Action*
1	link	pk	GET	https://172.19.160.198:8000/detail15
2	link	pk	GET	https://172.19.160.198:8000/detail14
3	link	pk	GET	https://172.19.160.198:8000/detail13
4	link	pk	GET	https://172.19.160.198:8000/detail12
5	link	pk	GET	https://172.19.160.198:8000/detail11
6	link	pk	GET	https://172.19.160.198:8000/detail10
7	link	pk	GET	https://172.19.160.198:8000/detail9
8	link	pk	GET	https://172.19.160.198:8000/detail6
9	link	pk	GET	https://172.19.160.198:8000/detail7
10	link	pk	GET	https://172.19.160.198:8000/detail8
11	link	pk	GET	https://172.19.160.198:8000/detail5

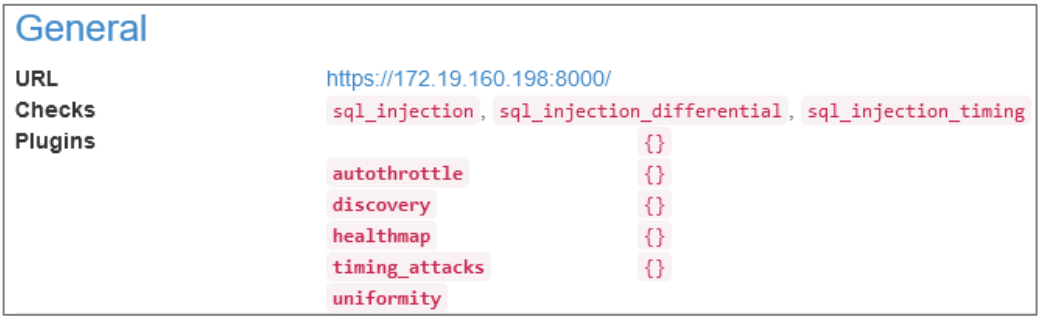
No	*Vector type*	*Input name*	*HTTP method*	*Action*
12	link	pk	GET	https://172.19.160.198:8000/detail4
13	link	pk	GET	https://172.19.160.198:8000/detail3
14	link	pk	GET	https://172.19.160.198:8000/detail1
15	link	pk	GET	https://172.19.160.198:8000/detail2

Berdasarkan tabel tersebut, kode-kode SQLi diinjeksi ke dalam *input* dengan nama pk (*Primary Key*) menggunakan metode GET. Pada kolom *action*, terdapat rincian URL yang *vulnerable* terhadap serangan SQLi. URL tersebut merupakan laman yang menampilkan detail dari isian *chat sender* pada formulir berikut.



Gambar 4.42. Tangkapan Layar Formulir pada *Webste* Uji Coba

Untuk melakukan serangan SQLi, Arachni memerlukan waktu *runtime* selama 01:44:22. Simulasi serangan Arachni menggunakan profil *SQL Injection* dengan parameter konfigurasi sebagai berikut.



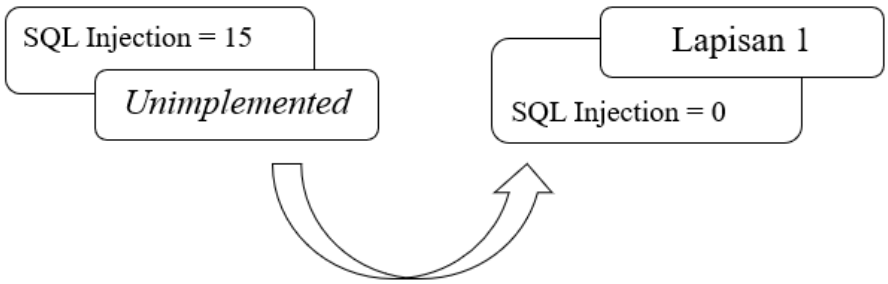
Gambar 4.43. Parameter Konfigurasi Arachni Ketika Melakukan Serangan SQLi

2) Tahapan *Implemented*

Metode mitigasi diimplementasikan untuk memitigasi serangan SQLi. Pada tahap *unimplemented*, terdapat 15 celah keamanan SQLi yang ditemukan. Oleh karena itu, pada tahap *implemented*, metode mitigasi diimplementasikan untuk mengatasi serangan SQLi tersebut. Berikut ini adalah hasil implementasi metode mitigasi berdasarkan urutan lapisan mitigasi.

Lapisan Pertama

Untuk memitigasi serangan SQLi pada lapisan pertama, *core rule set* yang digunakan adalah REQUEST-942-APPLICATION-ATTACK-SQLI. Setelah *rule* ini diimplementasikan, tidak ada serangan SQLi yang dapat dilakukan. Gambaran hasil serangan SQLi yang dapat dilakukan adalah sebagai berikut.



Gambar 4.44. Hasil Implementasi Mitigasi Serangan SQLi pada Lapisan Pertama

Sesuai dengan gambar tersebut, serangan SQLi tidak dapat dilakukan setelah lapisan pertama diterapkan. Hal ini terjadi pada Arachni dan ZAP, keduanya tidak dapat melakukan serangan SQLi. Berikut ini adalah cuplikan *core rule set* OWASP ModSecurity yang digunakan untuk memitigasi serangan SQLi.

```
1.  #
2.  # -= Paranoia Level 0 (empty) =- (apply unconditionally)
3.  #
4.
5.  SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1"
    "id:942011,phase:1,pass,nolog,skipAfter:END-REQUEST-942-APPLICATION-ATTACK-SQLI"
```

```
6.  SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1"
    "id:942012,phase:2,pass,nolog,skipAfter:END-REQUEST-942-APPLICATION-ATTACK-SQLI"
7.  #
8.  # -= Paranoia Level 1 (default) =- (apply only when tx.detection_paranoia_level
    is sufficiently high: 1 or higher)
9.  #
10.
11. #
12. # References:
13. #
14. # SQL Injection Knowledgebase (via @LightOS) -
15. # http://websec.ca/kb/sql_injection
16. #
17. # SQLi Filter Evasion Cheat Sheet -
18. # http://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/
19. #
20. # SQL Injection Cheat Sheet -
21. # http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
22. #
23. # SQLMap's Tamper Scripts (for evasions)
24. # https://svn.sqlmap.org/sqlmap/trunk/sqlmap/tamper/
25. #
26.
27. #
28. # -=[ LibInjection Check ]=-
29. #
30. # There is a stricter sibling of this rule at 942101. It covers REQUEST_BASENAME.
31. #
32. # Ref: https://libinjection.client9.com/
33. #
34. SecRule
    REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|REQUEST_HEADERS:Us
    er-Agent|REQUEST_HEADERS:Referer|ARGS_NAMES|ARGS|XML:/* "@detectSQLi" \
35.     "id:942100,\
36.     phase:2,\
37.     block,\
38.     capture,\
39.     t:none,t:utf8toUnicode,t:urlDecodeUni,t:removeNulls,\
40.     msg:'SQL Injection Attack Detected via libinjection',\
41.     logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}:
    %{MATCHED_VAR}',\
42.     tag:'application-multi',\
43.     tag:'language-multi',\
44.     tag:'platform-multi',\
45.     tag:'attack-sqli',\
46.     tag:'paranoia-level/1',\
47.     tag:'OWASP_CRS',\
48.     tag:'capec/1000/152/248/66',\
49.     tag:'PCI/6.5.2',\
50.     ver:'OWASP_CRS/4.0.0-rc1',\
51.     severity:'CRITICAL',\
52.     multiMatch,\
53.     setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}',\
54.     setvar:'tx.sql_injection_score=+%{tx.critical_anomaly_score}'"
55.
56.
```

```
57. #
58. # -=[ Detect DB Names ]=-
59. #
60. # Regular expression generated from util/regexp-assemble/data/942140.data.
61. # To update the regular expression run the following shell script
62. # (consult util/regexp-assemble/README.md for details):
63. #   util/regexp-assemble/regexp-assemble.py update 942140
64. #
65. SecRule
    REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XM
    L:/* "@rx
    (?i)\b(?:(?:m(?:s(?:ys(?:ac(?:cess(?:objects|storage|xml)|es)|(?:relationship|obj
    ect|querie)s|modules2?)|db)|aster\.\.sysdatabases|ysql\.db)|pg_(?:catalog|toast)|
    information_schema|northwind|tempdb)\b|s(?:(?:ys(?:\.database_name|aux)|qlite(?:_
    temp)?_master)\b|chema(?:_name\b|\W*\())|d(?:atabas|b_nam)e\W*\()" \
66.     "id:942140,\
67.     phase:2,\
68.     block,\
69.     capture,\
70.     t:none,t:urlDecodeUni,\
71.     msg:'SQL Injection Attack: Common DB Names Detected',\
72.     logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}:
    %{MATCHED_VAR}',\
73.     tag:'application-multi',\
74.     tag:'language-multi',\
75.     tag:'platform-multi',\
76.     tag:'attack-sqli',\
77.     tag:'paranoia-level/1',\
78.     tag:'OWASP_CRS',\
79.     tag:'capec/1000/152/248/66',\
80.     tag:'PCI/6.5.2',\
81.     ctl:auditLogParts=+E,\
82.     ver:'OWASP_CRS/4.0.0-rc1',\
83.     severity:'CRITICAL',\
84.     setvar:'tx.sql_injection_score=+%{tx.critical_anomaly_score}',\
85.     setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
86.
87.
88. #
89. # -=[ SQL Function Names ]=-
90. #
91. # This rule is a less stricter sibling of 942150.
92. #
93. # Regexp generated from util/regexp-assemble/data/942151.data using
    Regexp::Assemble.
94. # To rebuild the regexp:
95. #   cd util/regexp-assemble
96. #   ./regexp-assemble.py 942151
97. #
98. SecRule
    REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XM
    L:/* "@rx
    (?i)\b(?:s(?:q(?:lite_(?:compileoption_use|source_i)d|rt)|t(?:d(?:dev_(?:sam|po)p
    )?|r(?:_to_date|cmp))|ub(?:str(?:ing(?:_index)?)?|(?:dat|tim)e)|e(?:ssion_user|c_
    to_time)|ys(?:tem_user|date)|ha[12]?|oundex|chema|pace|in)|c(?:o(?:n(?:v(?:ert(?:
    _tz)?)?|cat(?:_ws)?|nection_id)|(?:mpres)?s|ercibility|llation|alesce|t)|ur(?:ren
```

```
    t_(?:time(?:stamp)?|date|user)|(?:dat|tim)e)|ha(?:racte)?r_length|iel(?:ing)?|r32
    )|i(?:s(?:_(?:ipv(?:4(?:_(?:compat|mapped))?|6)|n(?:ot(?:_null)?|ull)|(?:free|use
    d)_lock)|null)|n(?:et(?:6_(?:aton|ntoa)|_(?:aton|ntoa))|s(?:ert|tr)|terval)|fnull
    )|l(?:o(?:ca(?:ltimestamp|te)|g(?:10|2)|ad_file|wer)|i(?:kel(?:ihood|y)|nestring)
    |ast_(?:inser_id|day)|e(?:as|f)t|case|trim|pad)|d(?:a(?:t(?:e(?:_(?:format|add|su
    b)|diff)|abase)|y(?:of(?:month|week|year)|name))|e(?:s_(?:de|en)crypt|grees|code)
    |count|ump)|u(?:n(?:compress(?:ed_length)?|ix_timestamp|likely|hex)|tc_(?:time(?:
    stamp)?|date)|uid(?:_short)?|pdatexml|case)|t(?:ime(?:_(?:format|to_sec)|stamp(?:
    diff|add)?|diff)|o(?:(?:second|day)s|_base64|n?char)|r(?:uncate|im))|m(?:a(?:ke(?
    :_set|date)|ster_pos_wait)|ulti(?:po(?:lygon|int)|linestring)|i(?:crosecon)?d|ont
    hname|d5)|g(?:e(?:t_(?:format|lock)|ometrycollection)|(?:r(?:oup_conca|eates)|tid
    _subse)t)|p(?:o(?:(?:siti|lyg)on|w)|eriod_(?:diff|add)|rocedure_analyse|g_sleep)|
    a(?:s(?:cii(?:str)?|in)|es_(?:de|en)crypt|dd(?:dat|tim)e|tan2?)|f(?:rom_(?:unixti
    me|base64|days)|i(?:el|n)d_in_set|ound_rows)|e(?:x(?:tract(?:value)?|p(?:ort_set)
    ?)|nc(?:rypt|ode)|lt)|b(?:i(?:t_(?:length|count|x?or|and)|n_to_num)|enchmark)|r(?
    :a(?:wtohex|dians|nd)|elease_lock|ow_count|trim|pad)|o(?:(?:ld_passwo)?rd|ct(?:et
    _length)?)|we(?:ek(?:ofyear|day)|ight_string)|json(?:_(?:object|array))?|n(?:ame_
    const|ot_in|ullif)|var(?:_(?:sam|po)p|iance)|qu(?:arter|ote)|hex(?:toraw)?|yearwe
    ek|xmltype)\W*\(" \
99.     "id:942151,\
100.          phase:2,\
101.          block,\
102.          capture,\
103.          t:none,t:urlDecodeUni,t:lowercase,\
104.          msg:'SQL Injection Attack',\
105.          logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}:
    %{MATCHED_VAR}',\
106.          tag:'application-multi',\
107.          tag:'language-multi',\
108.          tag:'platform-multi',\
109.          tag:'attack-sqli',\
110.          tag:'OWASP_CRS',\
111.          tag:'capec/1000/152/248/66',\
112.          tag:'PCI/6.5.2',\
113.          tag:'paranoia-level/1',\
114.          ctl:auditLogParts=+E,\
115.          ver:'OWASP_CRS/4.0.0-rc1',\
116.          severity:'CRITICAL',\
117.          setvar:'tx.sql_injection_score=+%{tx.critical_anomaly_score}',\
118.          setvar:'tx.anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
```

Gambar 4.45. *Core Rule Set* Serangan SQLi

Sesuai dengan hasil uji coba serangan yang telah dilakukan, serangan SQLi dapat dimitigasi pada lapisan pertama. Dari 15 serangan SQLi pada tahap *unimplemented*, tak ada lagi satupun serangan yang dapat dilakukan setelah lapisan pertama metode mitigasi diimplementasikan. Dengan demikian, uji coba mitigasi serangan SQLi tidak perlu berlanjut atau mengaktifkan lapisan-lapisan berikutnya. Penjelasan berikutnya akan terkait dengan metode mitigasi serangan RCE.

4.2.2.3. Teknik Serangan *Remote Code Execution* (RCE)

Implementasi dan pengujian metode mitigasi *Remote Code Execution* memiliki beberapa perbedaan *minor* dengan serangan XSS dan SQLi. Untuk menguji serangan RCE, peneliti membangun *isolated environment* atau lingkungan yang terisolasi sebelum melakukan uji coba serangan RCE. Hal ini dilakukan untuk menghindari *crash* atau *accident* baik pada tingkat aplikasi, *website server*, atau bahkan pada tingkat sistem operasi saat menerima serangan RCE.

Dalam hal ini, kode-kode serangan RCE dapat sangat berbahaya karena sangat mungkin aplikasi penyerang mengirimkan kode-kode berbahaya seperti perintah untuk menghapus folder, berkas, atau bahkan *home folder*. Isolasi ini dilakukan untuk menghindari tersebut, sehingga uji coba implementasi serangan RCE tetap dapat dilakukan. Untuk melakukan tindakan isolasi RCE ini, terdapat dua hal yang dilakukan sebagai berikut:

1) Sebelum menerima *input* dari formulir serangan, terdapat sanitasi yang dilakukan pada sisi pemroses formulir seperti terlampir;
2) Dalam penerapan serangan Arachni, peneliti melakukan peninjauan atau *reviewing* terhadap kode serangan RCE yang dikirimkan.

Terkait dengan poin pertama di atas, *commands* RCE yang diisolasi adalah rm, mb, cp, :(){:|:&};:, wget, mkfs, dd, git, dan zip. Rincian kode isolasi yang diterapkan pada pemroses formulir adalah sebagai berikut.

```
1.      def get(self, request, *args, **kwargs):
2.          dangerous_commands = [
3.              'rm', 'mv', 'cp', ':(){:|:&};:',
4.              'wget', 'mkfs', 'dd', 'git', 'zip' ]
5.          keyword  = request.GET.get('keyword')
6.          query    = request.GET.get('query')
7.          response = ''
```

```
8.          next_url = reverse_lazy(get_index_url(request))
9.          if query:
10.             queries  = []
11.             _queries = [q.lower() for q in query.split(';')]
12.             for q in _queries:
13.                 passed = True
14.                 for c in dangerous_commands:
15.                     if not q.find(c) == -1:
16.                         passed = False
17.                 if passed:
18.                     queries.append(q)
19.             queries = ";".join(queries)
```

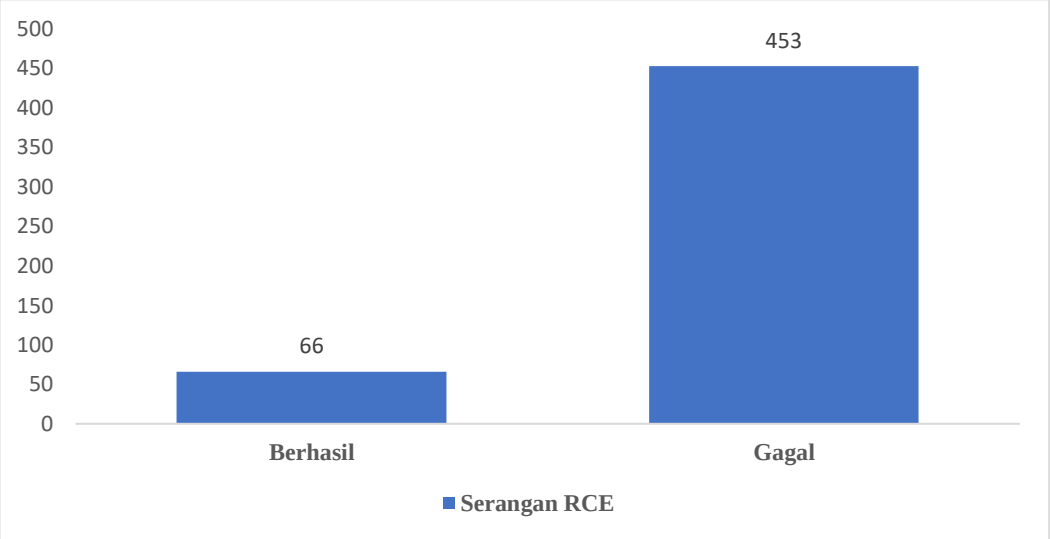
Gambar 4.46. Kode Isolasi Serangan RCE

Selain itu, pengujian serangan RCE juga dilakukan berdasarkan vektor serangan, tidak hanya berdasarkan URL. Terdapat 16 laman yang di dalamnya memiliki formulir yang dapat digunakan untuk mengeksploitasi serangan RCE. Uji coba serangan RCE berbasis URL cukup riskan karena juga berpotensi mendistorsi uji coba metode mitigasi serangan RCE. Terkait dengan uji coba simulasi serangan dan implementasi metode mitigasi serangan RCE dijelaskan sebagai berikut.

1) Tahapan *Unimplemented*

Berdasarkan uji coba yang telah dilakukan, Arachni dan ZAP mendapatkan jumlah celah keamanan RCE yang berbeda. Arachni dapat melakukan 66 serangan RCE, dan ZAP dapat melakukan 10 serangan RCE. Perbedaan jumlah keberhasilan serangan ini terjadi karena metode yang digunakan Arachni dan ZAP juga berbeda. Seperti yang telah disinggung sebelumnya, serangan Arachni tidak dilakukan secara langsung, sementara ZAP secara langsung. Vektor serangan Arachni diekspor menjadi dokumen format CSV, lalu setiap baris vektor serangan ditinjau terlebih dahulu. Arachni dan profil serangannya dikembangkan oleh kontributor komunitas dengan karakteristik yang berbeda-beda. Profil tersebut bisa saja mengandung vektor serangan RCE yang dapat mengganggu proses uji coba.

Setelah vektor serangan diekspor menjadi CSV, tahap selanjutnya adalah mengirimkan vektor serangan menggunakan *script* Python. Berdasarkan *log* vektor serangan, terdapat 512 serangan RCE yang dikirimkan ke *website* uji coba. Dari 512 serangan tersebut, sebanyak 66 dari 512 serangan tersebut sukses tereksekusi, sementara 453 vektor serangan gagal terseksekusi (grafik 4.2)



Grafik 4.49. Rekapitulasi Serangan RCE Arachni

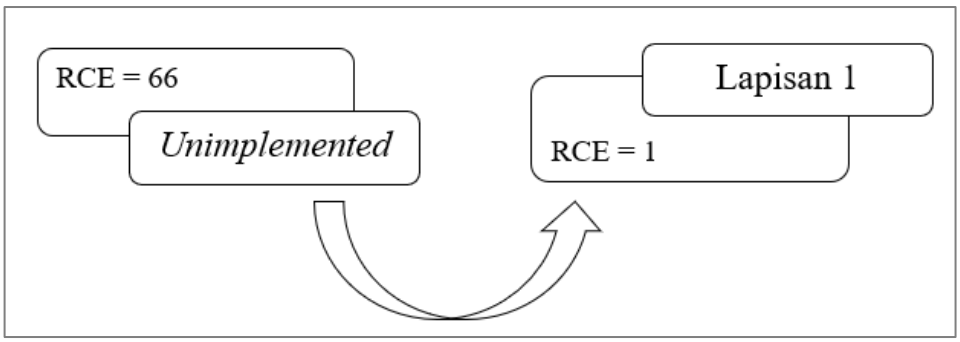
2) Tahapan *Implemented*

Berdasarkan uji coba serangan yang telah dilakukan pada tahapan *implemented*, serangan RCE Arachni berhasil dimitigasi oleh lapisan keempat, yaitu *data sanitizer*. Sementara itu, serangan RCE oleh ZAP dapat dimitigasi oleh lapisan pertama. Implementasi masing-masing lapisan dijelaskan sebagai berikut.

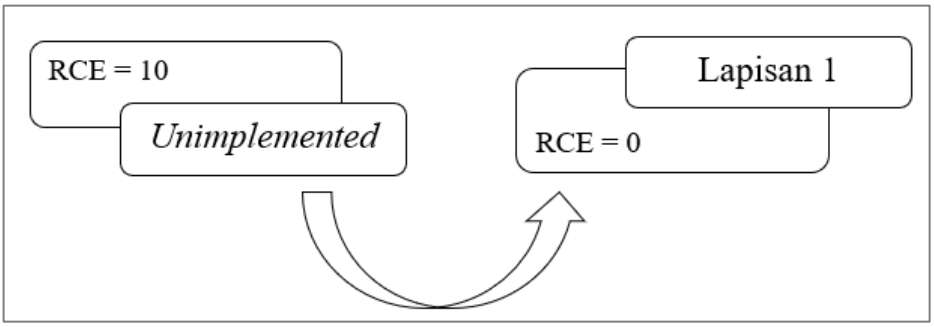
a) Lapisan Pertama (OWASP *ModSecurity*)

Untuk memitigasi serangan RCE, *core rule set* yang diaktifkan adalah REQUEST-932-APPLICATION-ATTACK-RCE.conf. Untuk mengaktifkan *rule* RCE ini, data pendukung RCE juga harus diaktifkan. Data pendukung yang diaktifkan

adalah *unix-shell.data*, *windows-powershell-commands.data* dan *restricted-upload .data*. Berdasarkan namanya, data pendukung tersebut berisi daftar perintah yang dapat dieksekusi oleh sistem operasi seperti Linux, Windows, dan seterusnya. Setelah *rules* tersebut diaktifkan, sebanyak 65 dari 66 serangan Arachni berhasil dimitigasi sehingga hanya menyisakan satu serangan saja. Sementara itu, semua serangan ZAP berhasil dimitigasi oleh lapisan pertama.



Gambar 4.47. Hasil Implementasi Mitigasi Serangan RCE Arachni Lapisan Pertama



Gambar 4.48. Hasil Implementasi Mitigasi Serangan RCE ZAP Lapisan Pertama

Cuplikan *core rule set* OWASP ModSecurity yang digunakan memitigasi serangan RCE adalah sebagai berikut. Data pendukung RCE harus disertakan atau diaktifkan sebelum *rule* ini diaktifkan. Jika tidak, *apache2* tidak akan dapat dimulai, karena terdapat *log error* pada *core rule set*.

```
1.  #
2.  # -= Paranoia Level 0 (empty) =- (apply unconditionally)
3.  SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1"
    "id:932011,phase:1,pass,nolog,skipAfter:END-REQUEST-932-APPLICATION-ATTACK-RCE"
```

```
4.  SecRule TX:DETECTION_PARANOIA_LEVEL "@lt 1"
    "id:932012,phase:2,pass,nolog,skipAfter:END-REQUEST-932-APPLICATION-ATTACK-RCE"
5.  #
6.  # -= Paranoia Level 1 (default) =- (apply only when tx.detection_paranoia_level
    is sufficiently high: 1 or higher)
7.  # [ Unix command injection ]
8.  #
9.  # This rule detects Unix command injections.
10. # A command injection takes a form such as:
11. #
12. #   foo.jpg;uname -a
13. #   foo.jpg||uname -a
14. #
15. # The vulnerability exists when an application executes a shell command
16. # without proper input escaping/validation.
17. #
18. # This rule is also triggered by an Oracle WebLogic Remote Command Execution
    exploit:
19. # [ Oracle WebLogic vulnerability CVE-2017-10271 - Exploit tested:
    https://www.exploit-db.com/exploits/43458 ]
20. #
21. # To prevent false positives, we look for a 'starting sequence' that
22. # precedes a command in shell syntax, such as: ; | & $( ` <( >(
23. # Anatomy of the regexp with examples of patterns caught:
24. #
25. # 1. Starting tokens
26. #
27. # ;         ;ifconfig
28. # \{        {ifconfig}
29. # \|        |ifconfig
30. # \|\|      ||ifconfig
31. # &         &ifconfig
32. # &&        &&ifconfig
33. # \n        ;\nifconfig
34. # \r        ;\rifconfig
35. # \$\(      $(ifconfig)
36. # $\(\(     $((ifconfig))
37. # `         `ifconfig`
38. # \${       ${ifconfig}
39. # <\(       <( ifconfig )
40. # >\(       >( ifconfig )
41. # \(\s*\)  a() ( ifconfig; ); a
42. #
43. # 2. Command prefixes
44. #
45. # {         { ifconfig }
46. # \s*\(\s* ( ifconfig )
47. # \w+=(?:[^\s]*|\$.*|\$.*|<.*|>.*|\'.*\'|\".*\")\s+   VARNAME=xyz ifconfig
48. # !\s*      ! ifconfig
49. # \$        $ifconfig
50. #
51. # 3. Quoting
52. #
53. # '         'ifconfig'
54. # \"        "ifconfig"
55. #
56. # 4. Paths
57. #
58. # [\?\*\[\]\(\)\-\|+\w'\"\./\x5c]+/   /sbin/ifconfig, /s?in/./ifconfig, /s[a-
    b]in/ifconfig etc.
59. # This rule is case-sensitive to prevent FP ("Cat" vs. "cat").
60. ……
```

Gambar 4.49. *Core Rule Set* Serangan RCE

b) Lapisan Kedua (*HTTP Middleware)*

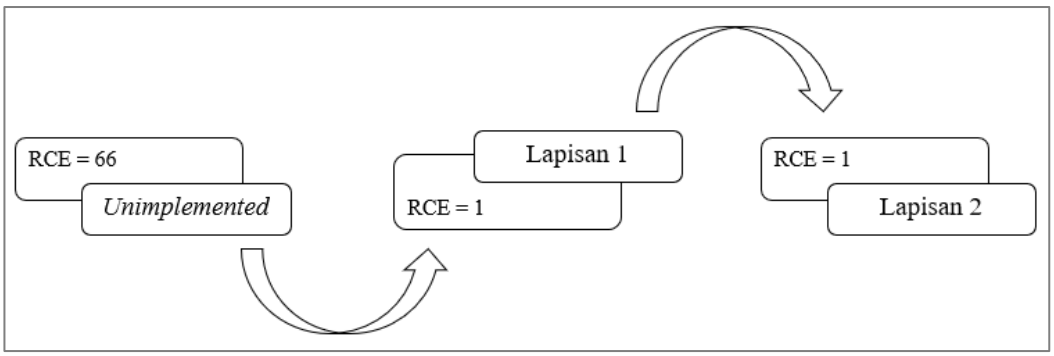
Sama dengan serangan XSS dan SQLi, metode deteksi serangan diletakan di HTTP *Middleware*. Pada lapisan ini, metode deteksi menandai atau memberikan *label payload* pada *request* yang tergolong Vektor Serangan. Jika label yang diberika adalah *payload*, aplikasi mengirimkan kode status HTTP 404. Berikut ini merupakan implementasi lapisan kedua dalam kode Python.

```
1.  import os
2.  from django.http import Http404
3.  from django.conf import settings
4.  from aurora.backend.library.mlsm.xss.classifier import XSSClassifier
5.  from aurora.backend.library.mlsm.rce.classifier import RCEClassifier
6.  from aurora.backend.library.mlsm.sqli.classifier import SQLiClassifier
7.
8.
9.  class MSLMMiddleware(object):
10.
11.     def __init__(self, get_response):
12.         self.get_response = get_response
13.
14.
15.     def __call__(self, request):
16.         response = self.get_response(request)
17.         return self.process_request(request, response)
18.
19.
20.     def process_request(self, request, response):
21.         GET_DATA  = request.get_full_path()
22.         POST_DATA = request.POST
23.         if settings.LAYER_2_MIDLEWARE == True:
24.             # Cross Site Scripting Detection
25.             # Remote Code Execution Detection
26.             # SQL Injection Detection
27.             classifiers = {
28.                 'xss'   : XSSClassifier(),
29.                 'rce'   : RCEClassifier(),
30.                 'sqli'  : SQLiClassifier()
31.             }
32.             for data in [GET_DATA, POST_DATA]:
33.                 for name, classifier in classifiers.items():
34.                     label = classifier.classify(data)
35.                     if label == 'payload':
36.                         raise Http404()
37.         return response
```

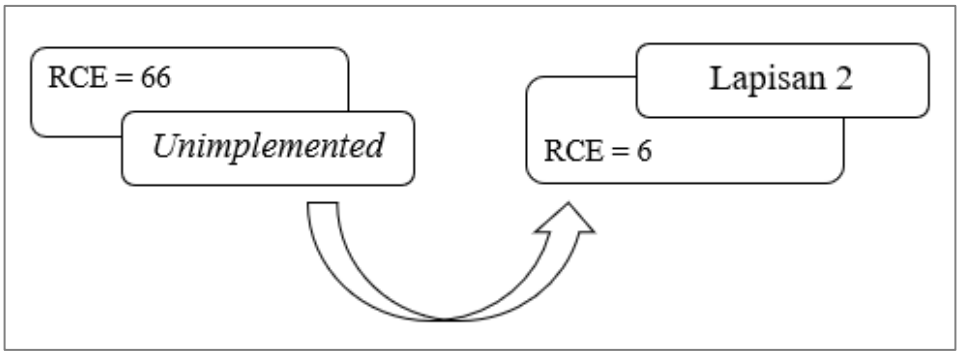
Gambar 4.50. Kode Python Mengaktifkan Metode Deteksi RCE pada *Middleware*

Setelah lapisan kedua di atas diimplementasikan, serangan RCE ternyata masih dapat dilakukan. Karena lapisan ini terkait dengan metode deteksi yang telah dihasilkan, pengujian juga dilakukan untuk memastikan bahwa metode deteksi yang

dapat melakukan deteksi. Apabila pengujian secara *standalone* tidak dilakukan, dikhawatirkan skpetis penelitian akan muncul dan tingkat akurasi metode deteksi yang telah dihasilkan juga akan dipertanyakan. Dalam hal ini, berdasarkan pengujian secara *standalone*, metode deteksi dapat mendeteksi 60 dari 66 serangan RCE dapat dideteksi metode deteksi. Berikut ini adalah gambaran hasil implementasi lapisan kedua dengan skema kolaboratif dan *standalone* dalam memitigasi serangan RCE.



Gambar 4.51. Hasil Implementasi Mitigasi RCE pada *Middleware (Integrative)*

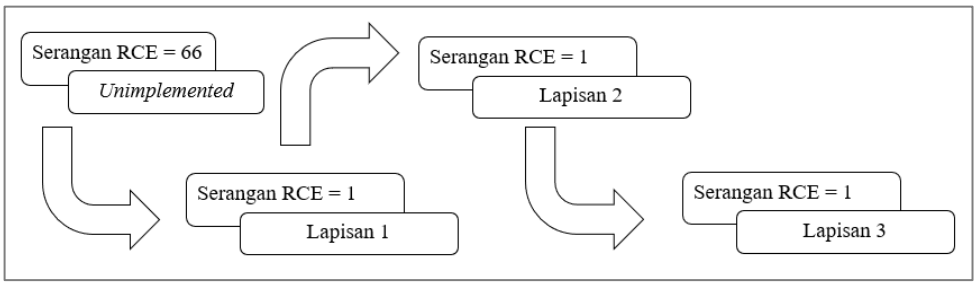


Gambar 4.52. Hasil Implementasi Deteksi RCE pada *Middleware* (*Standalone*)

Seperti yang telah tertera pada gambar tersebut, lapisan kedua skema kolaboratif belum mampu memitigasi serangan RCE. Hal yang sama juga terjadi pada skema *standalone*. Terdapat 6 serangan RCE yang masih dapat dilakukan pada lapisan kedua *standalone*. Namun demikian, lapisan kedua *standalone* tersebut tetap berhasil mendeteksi dan memitigasi 60 serangan sehingga hanya tersisa 6 serangan yang dapat dilakukan pada lapisan kedua.

c) Lapisan Ketiga (*Template Engine)*

Karena lapisan kedua tidak dapat memitigasi serangan RCE, pengujian pada lapisan ketiga juga dilakukan. Untuk memitigasi serangan RCE, pengaturan *auto-escape* pada *template engine* Django diaktifkan. Namun demikian, meskipun lapisan ketiga telah diaktifkan, serangan RCE masih tidak dapat dimitigasi. Tersisa satu serangan RCE yang masih dapat dilakukan meskipun ketiga lapisan sudah diaktifkan. Oleh karena itu, lapisan keempat harus diimplementasikan.



Gambar 4.53. Hasil Implementasi Deteksi RCE pada Lapisan Ketiga

d) Lapisan Keempat (*Data Sanitizer)*

Berdasarkan hasil uji coba serangan yang telah dilakukan, serangan RCE tetap dapat dilakukan meskipun lapisan 1, 2, dan 3 diaktifkan. Oleh karena itu, lapisan keempat diaktifkan. *Data sanitizer* diaktifkan sebelum formulir diproses. Hal ini untuk memastikan bahwa *input* yang dikirimkan sesuai dengan aturan yang telah ditetapkan. Berikut ini adalah implementasi *data sanitizer* dalam bentuk kode Python untuk memitigasi serangan RCE.

```
1.  def get(self, request, *args, **kwargs):
2.      keyword  = request.GET.get('keyword')
3.      query    = request.GET.get('query')
4.      response = ''
5.      next_url = reverse_lazy(get_index_url(request))
6.      if query:
7.          queries  = []
8.          _queries = [q.lower() for q in query.split(';')]
9.          for q in _queries:
10.             passed = True
11.             for c in dangerous_commands:
```

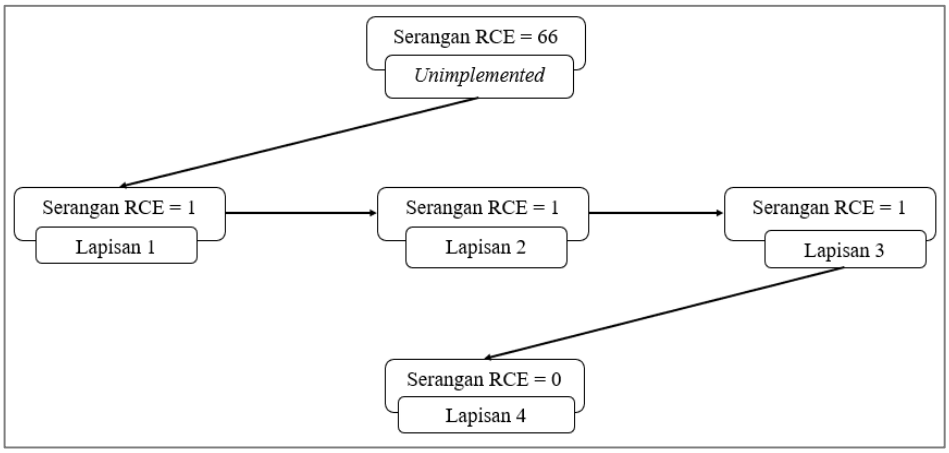
```
                if not q.find(c) == -1:
                    passed = False
            if passed:
                queries.append(q)
        queries = ";".join(queries)

        # implementation of data sanitizer
        if settings.LAYER_4_DATA_SANITIZER:
            sanitizers = {
                '&'  : '',
                ';'  : '',
                '|'  : '',
                '-'  : '',
                '$'  : '',
                '('  : '',
                ')'  : '',
                '`'  : '',
                '||' : '',
            }
            for key, value in sanitizers.items():
                queries = queries.replace(key, value)
        response = os.popen(queries).read()
        for sign in ['bytes', 'from']:
            if not sign in response.split(' '):
                raise Http404()

    if response:
        return HttpResponse(f"<div class='rce-reply'>{response}</div>")
    r = f'{next_url}?keyword={keyword}&query={query}&response={response}'
    return redirect(r)
```

Gambar 4.54. Kode Python untuk Melakukan Sanitasi pada *Input* Pengguna

Seperti yang tertera pada kode Python tersebut, sebelum *input* dari formulir diproses, data *input* disanitasi terlebih dahulu. Terkait dengan kode RCE yang dapat diterima adalah fungsi untuk melakukan *ping*, seperti konsep yang diterapkan pada DVWA. Jadi, setiap *input* yang mengandung karakter &, ;, |, -. $, (,), `, dan || akan dieliminasi atau diubah menjadi *blank*. Selain itu, sebelum respon dikirimkan, *output* RCE juga divalidasi. Dalam hal ini, apabila kata *bytes* dan *from* tidak ditemukan pada *output*, respon yang dikirimkan adalah status HTTP 404. Setelah lapisan ini diimplementasikan, serangan RCE tidak lagi dapat dilakukan.

Berpijak pada hasil implementasi lapisan keempat atau *data sanitizer* ini, akses terhadap fungsi-fungsi sistem operasi memang tidak dapat dihindari. Karena satu atau beberapa kebutuhan, fungsi-fungsi di dalam sistem operasi harus digunakan karena alasan efisiensi waktu. Namun demikian, sebelum menggunakan fitur ini,

apalagi jika fungsi ini dapat diakses pada cakupan *frontend*, setiap *input* yang masuk harus disanitasi terlebih dahulu, karena serangan RCE akan mengeksploitasi celah keamanan pada sisi sistem operasi. Dengan mengimplementasikan sanitasi data, potensi serangan RCE dapat dimitigasi secara efektif.



Gambar 4.55. Hasil Implementasi Mitigasi RCE Lapisan Pertama - Keempat

4.2.3. Relevansi dan Komparasi dengan Hasil Penelitian Sebelumnya

Untuk menghindari *missing links* atau rantai yang terputus pada pemaparan hasil dan pembahasan, serta untuk mempermudah identifikasi relevansi dan melakukan komparasi hasil penelitian ini dengan penelitian sebelumnya, penjelasan terkait relevansi dan komparasi penelitian dibuat tersendiri. Meskipun dijelaskan secara terpisah, penjelasan diupayakan tidak keluar dari konteks pembahasan. Dengan demikian, relevansi dan komparasi dengan penelitian sebelumnya tetap dapat disampaikan secara utuh dan komprehensif.

4.2.3.1. Metode Deteksi

1) *Cross Site Scripting* (XSS)

Penelitian terkait penggunaan SVM untuk mendeteksi serangan XSS dilakukan oleh Mokbal *et al*. Penelitiannya membandingkan tingkat akurasi algoritma

Decission Tree, CNN+SVM, *Random Forest*, dan NLP-SVM. Berdasarkan penelitian tersebut, SVM menjadi algoritma dengan tingkat akurasi tertinggi, yaitu mencapai 0,9944 [91]. Selain itu, Hermita juga menggunakan SVM untuk melakukan deteksi serangan XSS. Berdasarkan hasil penelitiannya, SVM dapat melakukan deteksi dengan tingkat akurasi sebesar 99,35% [92].

Buz *et al* [26] menggunakan *Convolutional Neural Network* (CNN) untuk melakukan deteksi serangan XSS dengan tingkat akurasi 0.99013. Jika dibandingkan dengan tingkat akurasi yang dihasilkan pada penelitian sebelumnya tersebut, tingkat akurasi pada penelitian ini tegolong lebih besar, karena mencapai 0,996546. Pada kasus yang berbeda, Prasetio *et al* menggunakan *Logistic Regression* dengan *hybrid feature*. Dengan jumlah dataset yang lebih sedikit dari penelitian ini, tingkat akurasi yang diperoleh mencapai 99,87% [16].

Berbeda dengan penelitian tersebut, *Logistic Regression* pada penelitian ini hanya mendapatkan tingkat akurasi sebesar 0,99471 dengan ToP mencapai 0,0008. Selain itu, salah satu dataset yang digunakan Prasetio merupakan dataset yang digunakan untuk melakukan pengujian kinerja metode deteksi. Sayangnya, Prasetio *et al* tidak mencantumkan ToP/*query*, juga tidak melakukan pengujian kinerja pada dataset yang berbeda. Tanpa informasi ToP/*query*, kecepatan metode deteksi juga tak dapat diketahui, apalagi penelitiannya juga memanfaatkan *hybrid feature*s atau fitur campuran (unigram, bigram, dan trigram) yang berpotensi mengkonsumsi *memory* lebih besar. Disamping itu, tidak dilakukan juga pengujian daya tahan deteksi menggunakan bantuan aplikasi penyerang. Oleh karena itu, pengujian yang dilakukannya menjadi kurang komprehensif.

2) *SQL Injection* (SQLi)

Jemal *et al* menghimpun 11 penelitian deteksi serangan SQLi menggunakan *machine learning*. Berdasarkan data yang dihimpunnya, tingkat akurasi yang didapatkan *Naïve Bayes* mencapai 97,6%, *Backpropagation Neural Network* 96,8%, *Neural Network Based Model 95%, Neural Network 98,6%, Decission Tree 83,7%, Neural Network Multi-Layer Feed Forward 66,67%, SVM* 96,23%, KNN 97%, dan *Neural Network* dengan teknik TbD-NNbR mencapai 99,23%.

Sementara itu, Kranthikumar dan Velusamy menggunakan REGEX *classifier* untuk mendeteksi serangan SQLi. Berdasarkan riset yang telah dilakukannya, tingkat akurasi REGEX mencapai 97% dengan 3 .98 detik waktu komputasional [30]. Dalam kaitannya dengan penelitian ini, tingkat akurasi metode deteksi yang dicapai pada penelitian ini mencapai 0,997713 atau 99.77% dengan ToP 0,001008. Dilihat dari urutannya, tingkat akurasi yang dicapai pada penelitian ini lebih besar dibandingkan penelitian-penelitian sebelumnya. Selain itu, setelah diujikan dengan dataset berbeda, dengan jumlah 33.514 baris data, metode deteksi pada penelitian ini mampu mendeteksi serangan SQLi dengan persentase 99,37%.

3) *Remote Code Execution* (RCE)

Berdasarkan *systematic literature* review yang dilakukan, kebanyakan metode deteksi serangan RCE dilakukan secara konvensional, tidak menggunakan *machine learning*. Sebagai contoh, penelitian Sharma dan Tomar mengajukan metode deteksi serangan RCE secara konvensional [93], tidak menggunakan *machine learning*. Sementara itu, penelitian ini telah menghasilkan metode deteksi dengan tingkat akurasi 0,987495 dengan ToP 0,004356. Riset terkait deteksi RCE jumlahnya

terbatas, sehingga penelitian ini tidak dapat menampilkan relevansi dan komparasi seperti teknik serangan lainnya. Dalam hal ini, penelitian berikutnya diharapkan dapat memberikan perbandingan pada deteksi serangan RCE.

4.2.3.2. Metode Mitigasi

1) *Cross Site Scripting* (XSS)

Metode mitigasi menggunakan OWASP ModSecurity (lapisan pertama pada *multi-layer security*) dilakukan oleh Akbar *et al*. Berdasarkan penelitian yang telah dilakukannya, terdapat serangan *Stored XSS* yang masih gagal dimitigasi [11]. Berbeda dengan penelitian tersebut, penelitian ini sukses memitigasi ketiga jenis serangan XSS, yaitu *stored, relfected,* dan DOM.

Beberapa penelitian terkait mitigasi serangan XSS juga pernah dilakukan oleh beberapa peneliti. Namun demikian, jumlah serangan XSS yang disediakan masih jauh dari jumlah yang disediakan pada penelitian ini. Sebagai contoh, metode mitigasi XSS menggunakan AES dilakukan oleh Putra *et al* [12]. Penelitian ini memitigasi celah keamanan yang terbatas, yaitu hanya dua celah keamanan XSS. Metode mitigasi XSS menggunakan OWASP ESAPI pernah dilakukan oleh Wijayarathna. Berdasarkan penelitiannya, terdapat 16 isu yang masih ditemukan [94].

2) *SQL Injection* (SQLi)

Implementasi metode mitigasi serangan SQLi menggunakan OWASP ModSecurity dilakukan oleh Akbar *et al*. Berdasarkan penelitian yang telah dilakukan, OWASP ModSecurity berhasil memitigasi 100% serangan SQLi [11]. Hasil serupa juga ditemukan pada penelitian yang dilakukan Prabowo dan Wahyu. Keduanya

menggunakan OWASP ModSecurity untuk memitigasi serangan SQLi. Berdasarkan hasil penelitian mereka, OWASP ModSecurity mampu memitigasi 100% serangan SQLi [95]. Merujuk pada hasil penelitian tersebut, serangan SQLi oleh ZAP juga dapat dimitigasi oleh OWASP ModSecurity. Namun demikian, serangan SQLi oleh Arachni baru dapat dimitigasi 100% oleh lapisan kedua. Rincian implementasi dan hasil mitigasi serangan SQLi dapat dilihat pada hasil dan pembahasan.

3) *Remote Code Execution* (RCE)

Metode mitigasi yang diterapkan pada penelitian ini mengacu pada beberapa kode sumber aplikasi web yang sering digunakan untuk meningkatkan kemampuan *ethical hacking*. Untuk memitigasi serangan RCE, DVWA (*Damn Vulnerable Web Application),* OWASP *Web Goat*, atau Bwapp (*buggy web application)* mengimplementasikan *data sanitizer* pada kode sumbernya [96], [97], dan [98].