Pendeteksian *Malware* pada Lingkungan Aplikasi Web dengan Kategorisasi Dokumen

Fransiskus Gusti Ngurah Dwika Setiawan, Royyana Muslim Ijtihadi, dan Hudan Studiawan
Jurusan Teknik Informatika, Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: fgn.dwikasetiawan@gmail.com, roy@if.its.ac.id, hudan@if.its.ac.id

***Abstrak*—Jumlah aplikasi berbasis web semakin bertambah seiring dengan perkembangan teknologi informasi. Dengan bertambahnya jumlah aplikasi web, serangan-serangan yang dilakukan terhadap aplikasi-aplikasi web tersebut juga meningkat. Salah satu jenis serangan yang marak dilakukan terhadap aplikasi web adalah penyisipan *malware* seperti *web shell* yang dapat memberikan akses bebas terhadap komputer server kepada penyerang. Dalam makalah ini, dijelaskan implementasi aplikasi yang menerapkan teknik kategorisasi dokumen untuk mendeteksi *malware* atau kode *malicious* khususnya jenis *web shell* dengan teknik kategorisasi dokumen. Proses kategorisasi dokumen meliputi praproses dan tokenisasi kode sumber, pembuatan model classifier *Multinomial Naive Bayes* dan *Decision Tree*, dan klasifikasi dokumen menggunakan classifier yang telah dibuat. Uji coba yang dilakukan terhadap 718 file kode sumber PHP menghasilkan tingkat precision dari 72% hingga 83% dan recall 83% hingga 97%.**

***Kata Kunci*—aplikasi web, deteksi *malware*, kategorisasi dokumen**

I. PENDAHULUAN

SEIRING berkembangnya teknologi informasi, jumlah aplikasi web yang tersebar di internet semakin meningkat. Salah satu hal yang memacu peningkatan tersebut adalah semakin maraknya perangkat-perangkat lunak yang memudahkan seseorang dalam membuat aplikasi web. Salah satu contoh perangkat lunak yang dapat mempermudah pembuatan aplikasi web tersebut adalah sistem manajemen konten *Wordpress*. Berkat perangkat-perangkat lunak tersebut, seseorang tidak perlu memiliki pengetahuan teknis yang tinggi untuk dapat membuat aplikasi web. Namun sayangnya beberapa pemilik aplikasi web tidak begitu memperhatikan atau paham aspek-aspek keamanan aplikasi dan jaringan. Hal ini memicu terjadinya serangan-serangan digital yang dilakukan terhadap aplikasi-aplikasi web yang tidak terproteksi.

Salah satu bentuk serangan yang dapat dilakukan adalah penyisipan file-file berbahaya atau *malicious* ke komputer penyedia aplikasi web. Contoh dari file berbahaya yang dapat disisipkan adalah skrip *web shell*. Apabila berhasil diunggah, skrip *web shell* dapat memberikan akses *shell* tidak sah kepada pengunggah skrip secara jarak jauh. Hal ini berpotensi merusak aplikasi web dan juga komputer server yang menyediakan aplikasi web tersebut.

Efek dari serangan berupa penyisipan file *malicious* dapat diatasi dengan memindai file-file pada aplikasi web untuk mencari dan menghapus file-file yang terdeteksi *malicious*. Salah satu metode pendeteksian file *malicious* yang pernah dilakukan adalah dengan menggunakan kategorisasi dokumen. Kategorisasi dokumen adalah suatu teknik untuk memetakan sebuah dokumen ke dalam satu atau beberapa kategori dengan menggunakan teknik *machine learning*. A. Shabtai et al. dalam penelitiannya[1] melakukan pendeteksian kode malicious dalam file-file biner dengan melakukan kategorisasi terhadap pola *OpCode*. *OpCode* adalah bagian dari instruksi bahasa mesin yang mendefinisikan operasi apa yang harus dilakukan.

Makalah ini membahas tentang penerapan teknik kategorisasi dokumen untuk mendeteksi *malware* di lingkungan aplikasi web. Makalah ini disusun dengan urutan sebagai berikut: pendahuluan, dasar teori, metode, perancangan sistem, uji coba dan evaluasi, kesimpulan, dan saran.

II. DASAR TEORI

A. Kategorisasi Dokumen

Kategorisasi teks adalah pekerjaan pemberian label kategori kepada sebuah dokumen teks [2]. Label kategori tersebut diambil dari kumpulan kategori-kategori yang telah ditentukan sebelumnya. Secara formal kategorisasi teks dapat didefinisikan sebagai kegiatan pemberian nilai *Boolean true* atau *false* kepada setiap pasangan dokumen-kategori $\langle d_i, c_j \rangle \in D \times C$, dengan $\boldsymbol{D}$ adalah himpunan semua dokumen dan $\boldsymbol{C}$ adalah himpunan semua kategori. Nilai *true* menyatakan bahwa dokumen $\boldsymbol{d_i}$ termasuk dalam kelas $\boldsymbol{c_j}$, sebaliknya nilai *false* menyatakan bahwa dokumen $\boldsymbol{d_i}$ tidak termasuk dalam kelas $\boldsymbol{c_j}$.

Untuk dapat melakukan kategorisasi dokumen secara otomatis, perlu diketahui fungsi target $\phi' : D \times C \rightarrow \{T, F\}$ yang mendeskripsikan bagaimana dokumen seharusnya diklasifikasi. Fungsi target tidak dapat diketahui secara pasti. Oleh karena itu, teknik *machine learning* diterapkan untuk mencari pendekatan fungsi target.

B. Vector Space Model

Vector Space Model adalah representasi matematis dari dokumen teks. Sebuah dokumen $\boldsymbol{d}$ yang di dalamnya terdapat $\boldsymbol{n}$ kata / *term* unik $\boldsymbol{t_1, t_2, t_3, ..., t_n}$ memiliki representasi vektor ($\boldsymbol{w_1, w_2, w_3, ..., w_n}$), dengan $\boldsymbol{w_i}$ adalah bobot atau nilai dari term $\boldsymbol{t_i}$ untuk dokumen $\boldsymbol{d}$. Representasi vektor ini biasa digunakan dalam sistem temu kembali informasi, namun dapat juga digunakan untuk kasus kategorisasi dokumen [2].

Dalam kasus kategorisasi dokumen, *classifier* menerima bentuk vektor dari dokumen untuk diklasifikasi. Ada banyak jenis bobot suatu term untuk sebuah dokumen. Beberapa contoh jenis bobot tersebut adalah bobot biner (0 atau 1), bobot TF (*Term Frequency*), dan bobot TF-IDF (*Term Frequency–Inverse Document Frequency*). Jenis bobot yang dipakai tergantung dari kasus penggunaan.

C. Term Frequency dan TF-IDF

Term Frequency dan TF-IDF (term frequency – inverse document frequency) adalah contoh jenis pembobotan term dalam kategorisasi dokumen dan temu kembali informasi [3]. *Term frequency* atau TF dari suatu *term* ***t*** untuk dokumen ***d*** didefinisikan sebagai jumlah kemunculan ***t*** dalam ***d***. Semakin panjang suatu dokumen, maka nilai TF masing-masing termnya cenderung semakin besar. Untuk mengurangi efek ini, seringkali digunakan bentuk lain dari pembobotan TF, seperti (1).

$$TF'_{t,d} = \begin{cases} 1 + log_{10}\ TF_{t,d} & TF_{t,d} > 0 \\ 0 & TF_{t,d} = 0 \end{cases} \quad (1)$$

$\boldsymbol{TF_{t,d}}$ adalah nilai TF untuk term ***t*** dan dokumen ***d***, sedangkan $\boldsymbol{TF'_{t,d}}$ adalah nilai $\boldsymbol{TF_{t,d}}$ dalam bentuk logaritma. Dengan menggunakan logaritma, dokumen panjang tidak akan memiliki bobot-bobot yang jauh lebih besar daripada dokumen pendek.

Selain pembobotan dengan TF, terdapat juga pembobotan dengan TF-IDF (Term Frequency – Inverse Document Frequency). Pembobotan TF-IDF mempertimbangkan seberapa unik suatu *term* dalam kumpulan dokumen. Pembobotan ini mengasumsikan bahwa suatu *term* yang muncul di banyak dokumen adalah *term* yang tidak penting. Semakin unik suatu *term*, maka *term* tersebut dianggap penting. Contohnya dalam kumpulan dokumen berbahasa Indonesia, *term* (kata) "di", "dan", "ke", "yang", "atau", dan "dan" muncul di banyak dokumen. Kata-kata tersebut tidak berkontribusi banyak dalam sistem pencarian dokumen atau kategorisasi dokumen. Oleh karena itu kata-kata tersebut harus diberi bobot yang lebih rendah daripada kata-kata yang lebih unik. Dari situ muncul persamaan pembobotan TF-IDF seperti yang ditunjukkan pada (2).

$$TFIDF'_{t,d} = \begin{cases} 1 + log_{10}\, TF_{t,d} \times N/df_t & TF_{t,d} > 0 \\ 0 & TF_{t,d} = 0 \end{cases} \quad (2)$$

N adalah jumlah dokumen, dan $\boldsymbol{df_t}$ adalah *document frequency* dari *term* ***t***, yakni jumlah dokumen yang mengandung *term* ***t***.

D. Naive Bayes

Naive Bayes merupakan salah satu algoritma *machine learning*. Algoritma ini menerapkan teorema Bayes untuk mengklasifikasi suatu data dengan mengasumsikan bahwa fitur-fitur dalam data tersebut tidak bergantung oleh satu sama lainnya (independen) [4]. Teorema Bayes menyatakan bahwa peluang dari sebuah sampel $\boldsymbol{d} = (\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n})$ termasuk dalam sebuah kelas ***c*** adalah:

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)} \quad (3)$$

Dengan mengasumsikan bahwa masing-masing fitur pada ***d*** tidak bergantung satu sama lain, maka persamaan (3) menjadi:

$$p(c|d) = \frac{p(c)}{p(d)} \prod_{i=1}^{n} p(x_i|c) \propto p(c) \prod_{i=1}^{n} p(x_i|c) \quad (4)$$

Dokumen ***d*** termasuk dalam kelas dengan nilai $\boldsymbol{p(c|d)}$ tertinggi untuk $\boldsymbol{c} \in \boldsymbol{C}$ dengan ***C*** adalah himpunan semua kelas yang mungkin untuk dokumen ***d***.

Salah satu bentuk dari algoritma *Naive Bayes* untuk mengklasifikasi dokumen adalah *Multinomial Naive Bayes* [5]. Dalam penerapan *Multinomial Naive Bayes* untuk klasifikasi dokumen, nilai $\boldsymbol{x_i}$, $\boldsymbol{i}$=1...$\boldsymbol{n}$ merupakan nilai numerik, maka dari itu $\boldsymbol{p(c|d)}$ didefinisikan seperti pada (5).

$$p(c|d) \propto p(c) \prod_{i=1}^{n} p^{x_i}(x_i|c) \quad (5)$$

Nilai $\boldsymbol{p(x|c)}$ merupakan frekuensi relatif term ***x*** dari dokumen-dokumen dalam kelas ***c***. Nilai tersebut didefinisikan seperti pada (6).

$$p(x|c) = \frac{T_{x,c}}{\sum_{x' \in X} T_{x',c}} \quad (6)$$

Dengan $\boldsymbol{T_{x,c}}$ adalah jumlah nilai TF atau TF-IDF dari *term* ***x*** dari semua dokumen dalam kelas ***c***, dan ***X*** adalah himpunan seluruh *term*.

Terkadang dalam suatu kelas tidak terdapat sama sekali *term* ***x***. Hal ini menyebabkan nilai $\boldsymbol{T_x}$ menjadi nol. Apabila nilai $\boldsymbol{T_x}$ = 0 dibiarkan, maka nilai $\boldsymbol{p(x|c)}$ pasti nol juga. Untuk mencegah hal ini, biasanya nilai $\boldsymbol{T_x}$ ditambah satu, sehingga (6) akan menjadi seperti (7).

$$p(x|c) = \frac{T_{x,c} + 1}{\sum_{x' \in X} T_{x',c} + 1} \quad (7)$$

Nilai-nilai $\boldsymbol{p(c)}$, $\boldsymbol{\log\ p(c)}$, $\boldsymbol{p(x_i \mid c)}$, $\boldsymbol{\log\ p(x_i \mid c)}$ untuk masing-masing kelas dihitung pada tahap pembangunan model atau training. Nilai-nilai tersebut dapat disimpan untuk mengklasifikasi data-data baru.

E. Decision Tree

Decision Tree adalah salah satu jenis algoritma *machine learning* yang lazim digunakan untuk klasifikasi maupun regresi [6]. Terdapat beberapa jenis algoritma Decision Tree. Beberapa di antaranya adalah: ID3, C4.5, dan CART.

Tujuan dari algoritma *Decision Tree* secara umum adalah memprediksi nilai dari variabel target ***y*** dari sampel ***x*** yang memiliki ***n*** fitur $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, ..., $\boldsymbol{x_n}$. Hal ini dilakukan dengan membuat sebuah *tree*. Pada algoritma CART, jenis *tree* yang dibuat adalah binary *tree* [7].

Masing-masing *node* internal pada *tree* merepresentasikan suatu dataset. Dataset tersebut kemudian dipecah ke dalam dua *node* baru di sebelah kanan dan kiri *node* sebelumnya. Pemecahan dataset tersebut dilakukan berdasarkan pengecekan nilai dari sebuah fitur $\boldsymbol{x_j}$. Apabila sampel $\boldsymbol{x^i}$ memiliki fitur $\boldsymbol{x^i_j}$ yang bernilai kurang dari sebuah nilai $\boldsymbol{s_j}$, maka sampel tersebut masuk ke *node* cabang kiri dari *node* sekarang. Apabila

sebaliknya, maka sampel tersebut masuk ke dalam *node* cabang kanan dari *node* sekarang. Proses tersebut dilakukan secara rekursif hingga terbentuk *leaf*. Sebuah *node* menjadi *leaf* apabila seluruh sampel dalam datasetnya memiliki nilai variabel target ***y*** yang sama, atau apabila jumlah sampel dalam datasetnya kurang dari nilai *threshold* tertentu.

Fitur yang dicek dan nilai s_j pada masing-masing *node* ditentukan dengan menggunakan aturan yang disebut *splitting rule*. Salah satu *splitting rule* yang lazim digunakan dalam algoritma CART adalah *gini splitting rule*. *Gini splitting rule* menentukan fitur yang digunakan untuk memecah dataset dengan cara menghitung perubahan nilai *gini impurity*. Pada kasus klasifikasi, nilai *gini impurity* untuk suatu *node* ***n*** dapat didefinisikan seperti pada (8).

$$I(n) = \sum_{y \in Y} f_{n,y}(1 - f_{n,y}) \quad (8)$$

Dengan $I(n)$ adalah nilai gini impurity dari node ***n***, ***Y*** adalah kumpulan nilai-nilai yang mungkin dari variabel target, $f_{n,y}$ adalah perbandingan jumlah sampel dengan kelas ***y*** pada *node* ***n*** dengan jumlah semua sampel pada *node* ***n***.

Perubahan nilai gini impurity dapat dihitung seperti pada (9).

$$\Delta I(n) = I(n) - I(n_l) - I(n_r) \quad (9)$$

$\Delta I(n)$ adalah perubahan nilai *gini impurity*, $I(n_l)$ adalah nilai *gini impurity* untuk *node* cabang kiri, dan $I(n_r)$ adalah nilai *gini impurity* untuk *node* cabang kanan. Nilai $\Delta I(n)$ dihitung untuk tiap-tiap fitur yang ada. Fitur dan nilai pemecah s_j yang dapat menghasilkan pemecahan dataset dengan nilai $\Delta I(n)$ terbesar akan digunakan untuk memecah dataset pada node ***n***.

Data baru dapat diklasifikasi dengan cara melakukan pengecekan nilai fitur dari data tersebut sesuai dengan *node -node* pada model *tree* yang dibentuk, mulai dari *node* awal sampai *leaf*. Nilai kelas ***y*** untuk data baru tersebut adalah kelas dengan jumlah sampel terbanyak pada *leaf* yang dicapai.

III. METODE

Ada tiga jenis proses yang perlu dilakukan untuk dapat mengklasifikasi apakah suatu file *malicious* atau tidak. Proses-proses tersebut adalah: praproses, *training*, dan klasifikasi.

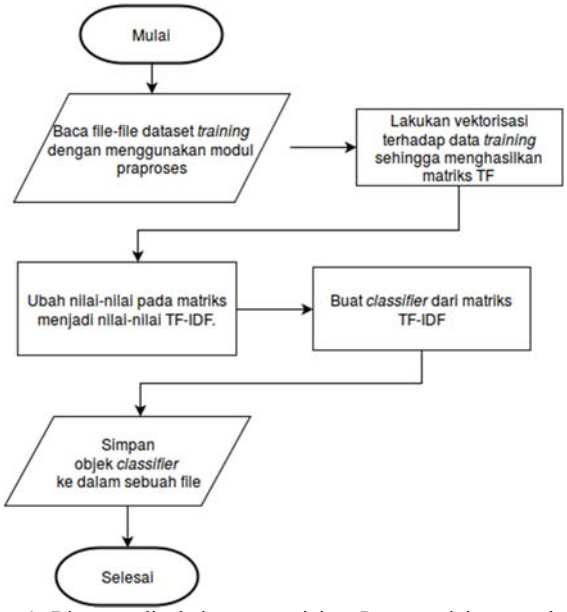
A. Praproses

Praproses bertujuan untuk mengambil potongan kode sumber PHP dari sebuah file, lalu melakukan tokenisasi terhadap potongan kode sumber tersebut. Kode sumber yang berupa sebuah *string* panjang akan dipecah menjadi kumpulan token-token. Token dalam konteks kode sumber PHP dapat berupa sebuah *keyword*, nama variabel, nama fungsi, konstanta, operator, *tag* PHP, dan sebagainya. Dari semua jenis-jenis token tersebut, yang akan digunakan dalam proses *training* dan klasifikasi adalah token nama variabel, nama fungsi, nama kelas, *keyword* eval dan echo, dan juga konstanta *string*. Untuk jenis token konstanta *string*, akan dilakukan pengelompokan seperti berikut:

1. *String* panjang (lebih dari nilai *threshold* tertentu) akan diubah menjadi token "LONG_NORMAL_STRING".
2. *String* dalam format base64 dengan panjang lebih dari nilai *threshold* tertentu akan diubah menjadi token "LONG_BASE64_STRING".
3. Token-token yang di dalamnya terdapat representasi karakter dalam bentuk heksadesimal (contoh: \x8F) akan diubah menjadi token "STRING_WITH_HEX_LITERAL".

Pengelompokan berdasarkan bentuk ini dilakukan untuk mengurangi jumlah fitur dokumen.

Kumpulan token-token tersebut merupakan luaran dari tahap praproses. Luaran tersebut kemudian akan digunakan pada tahap *training* dan klasifikasi.



Gambar 1. Diagram alir dari proses training. Proses training menghasilkan objek classifier dari data belajar masukan

jenis token konstanta *string*, akan dilakukan pengelompokan seperti berikut:

1. *String* panjang (lebih dari nilai *threshold* tertentu) akan diubah menjadi token "LONG_NORMAL_STRING".
2. *String* dalam format base64 dengan panjang lebih dari nilai *threshold* tertentu akan diubah menjadi token "LONG_BASE64_STRING".
3. Token-token yang di dalamnya terdapat representasi karakter dalam bentuk heksadesimal (contoh: \x8F) akan diubah menjadi token "STRING_WITH_HEX_LITERAL".

Pengelompokan berdasarkan bentuk ini dilakukan untuk mengurangi jumlah fitur dokumen.

Kumpulan token-token tersebut merupakan luaran dari tahap praproses. Luaran tersebut kemudian akan digunakan pada tahap *training* dan klasifikasi.

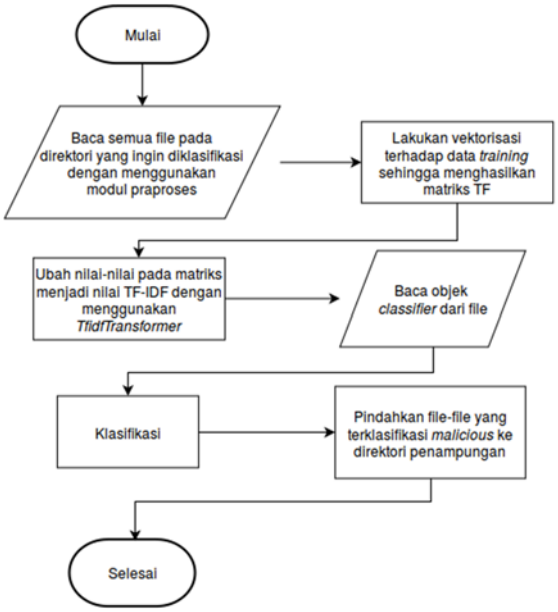
B. Training

Proses *training* bertujuan untuk menghasilkan model-model *classifier Multinomial Naive Bayes* dan *Decision Tree* dari data masukan. Diagram alir untuk proses ini dapat dilihat pada Gambar 1.

Proses *training* menerima data masukan berupa data belajar yang merupakan kumpulan kode sumber - kode sumber yang telah teridentifikasi sebagai kode sumber *malicious* atau

non-*malicious*. Sebelum menjadi masukan pada proses ini, data belajar tersebut akan dipraproses terlebih dahulu. Kemudian data tersebut akan diubah ke dalam bentuk vektor melalui proses vektorisasi. Proses vektorisasi menghasilkan matriks $\boldsymbol{M}$ berukuran $\boldsymbol{m} \times \boldsymbol{n}$, dengan $\boldsymbol{m}$ adalah jumlah dokumen dan $\boldsymbol{n}$ adalah jumlah token unik dari seluruh dokumen. Masing-masing baris pada $\boldsymbol{M}$ merepresentasikan sebuah dokumen, dan masing-masing kolomnya merepresentasikan sebuah token. Pada matriks $\boldsymbol{M}$, elemen $\boldsymbol{M_i}$, j berisi jumlah kemunculan token ke-$\boldsymbol{j}$ pada dokumen ke-$\boldsymbol{i}$. Nilai-nilai tersebut kemudian akan diubah menjadi nilai-nilai TF-IDF.

Setelah proses tokenisasi dan vektorisasi selesai, model-model *classifier* akan dibangun menggunakan algoritma *machine learning*. Algoritma *machine learning* yang digunakan dalam penelitian ini adalah *Multinomial Naive Bayes* dan *Decision Tree*. Objek-objek *classifier* yang dihasilkan akan disimpan masing-masing ke dalam sebuah file. File-file tersebut adalah luaran dari proses *training* dan akan digunakan dalam proses klasifikasi.



Gambar 2. Diagram alir dari proses klasifikasi. Proses ini menghasilkan prediksi-prediksi kategori untuk tiap-tiap file masukan dengan menggunakan objek-objek classifier hasil proses training.

C. Klasifikasi

Proses klasifikasi menerima file-file masukan dan objek-objek *classifier* hasil proses *training*. Hasil dari proses klasifikasi adalah kumpulan prediksi kategori dari masing-masing file masukan. Diagram alir proses klasifikasi dapat dilihat pada Gambar 2.

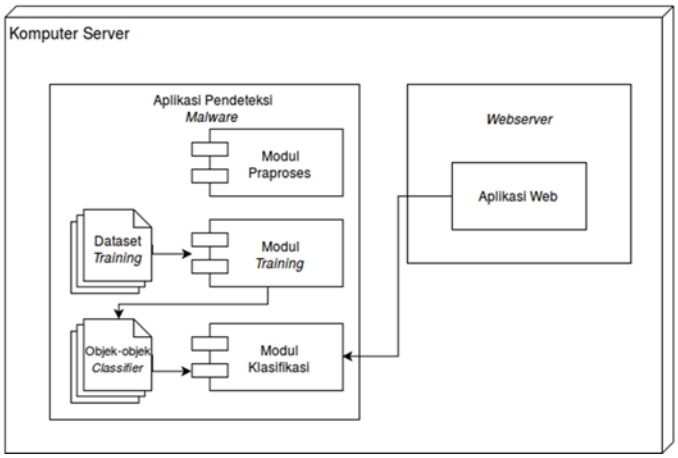
Proses klasifikasi dimulai dengan membaca file-file dari direktori yang telah ditentukan. File-file yang dibaca akan dipraproses terlebih dahulu. Kemudian akan dilakukan vektorisasi terhadap hasil praproses tersebut. Hasil vektorisasi berupa matriks $\boldsymbol{M}$ berukuran $\boldsymbol{m} \times \boldsymbol{n}$, dengan $\boldsymbol{m}$ adalah jumlah dokumen dan $\boldsymbol{n}$ adalah jumlah token unik dari seluruh dokumen. Nilai-nilai dari matriks $\boldsymbol{M}$ akan menjadi nilai TF-IDF. Matriks tersebut kemudian akan diklasifikasi dengan menggunakan model-model *classifier* yang telah dihasilkan oleh modul *training*.

IV. PERANCANGAN SISTEM

Aplikasi akan dipasang dalam komputer server yang berisi aplikasi web target. Arsitektur dari implementasi ini dapat dilihat pada Gambar 3.

Implementasi dari aplikasi pendeteksi malware ini akan memiliki tiga modul, yakni modul praproses, modul training, dan modul klasifikasi. Modul praproses akan menyediakan fungsi yang dapat digunakan oleh modul lain untuk mempraproses dokumen.

Modul *training* berfungsi untuk menjalankan proses *training* dengan masukan berupa dataset *training* yang telah disediakan sebelumnya. Modul klasifikasi dapat menerima kumpulan file-file dan melakukan klasifikasi terhadap file-file tersebut dengan menggunakan objek-objek *classifier* hasil dari modul *training*. Pada Gambar 3, ditunjukkan bahwa modul klasifikasi akan menerima file-file dari aplikasi web untuk diklasifikasi.



Gambar 3. Diagram arsitektur system. Di sini digambarkan lingkungan komputer server yang menyediakan sebuah aplikasi web. Aplikasi pendeteksi *malware* dipasang pada komputer tersebut, terpisah dari aplikasi web.

V. UJI COBA DAN EVALUASI

Penerapan teknik kategorisasi dokumen untuk pendeteksian *malware* yang telah diimplementasikan akan dievaluasi dengan menghitung nilai *precision* dan *recall* dari hasil uji coba klasifikasi terhadap dataset masukan. Nilai *precision* didefinisikan seperti pada (10).

$$P = \frac{TP}{TP + FP} \tag{10}$$

$\boldsymbol{P}$ adalah nilai *precision*, $\boldsymbol{TP}$ adalah jumlah *true positive* (jumlah file *malicious* yang terklasifikasi *malicious*), dan $\boldsymbol{FP}$ adalah jumlah *false positive* (jumlah file non-*malicious* yang terklasifikasi *malicious*). Nilai *recall* didefinisikan seperti pada (11).

$$R = \frac{TP}{TP + FN} \tag{11}$$

$\boldsymbol{R}$ adalah nilai *recall*, dan $\boldsymbol{FN}$ adalah jumlah *false negative* (jumlah file *malicious* yang terklasifikasi non-*malicious*). Uji coba dan perhitungan nilai *precision* dan *recall* akan dilakukan dalam beberapa skenario.

A. Skenario Uji Coba

Ada dua jenis uji coba yang akan dilakukan: uji coba tahap *training* dan uji coba tahap *testing*. Masing-masing jenis uji coba tersebut dilakukan dalam dua skenario, masing-masingnya menggunakan *classifier* yang berbeda. Skenario – skenario tersebut dapat dilihat pada Tabel 1.

Pada uji coba tahap *training*, sebuah model *classifier* akan dibuat dari dataset *training*. Hasil model *classifier* tersebut akan diuji coba untuk mengklasifikasi kembali dataset *training*. Sedangkan pada uji coba tahap *testing*, dataset *testing* akan diklasifikasi dengan model *classifier* yang telah dibangun pada uji coba tahap *training*.

Tabel 1. Rincian skenario-skenario uji coba.

No	Jenis Uji Coba	Dataset Masukan	Model *Classifier*
1	Uji coba tahap *training*	Dataset *training*	Multinomial Naive Bayes
2	Uji coba tahap *training*	Dataset *training*	Decision Tree
3	Uji coba tahap *testing*	Dataset *testing*	Multinomial Naive Bayes
4	Uji coba tahap *testing*	Dataset *testing*	Decision Tree

B. Deskripsi Data Uji Coba

Ada dua jenis dataset yang akan digunakan untuk uji coba, yakni dataset *training* dan dataset *testing*. Dataset training terdiri dari 427 kode sumber PHP *malicious* dan 500 kode sumber PHP non-*malicious*, dan dataset *testing* terdiri dari 218 kode sumber PHP *malicious* dan 500 kode sumber PHP non-*malicious*. Dataset *training* akan digunakan dalam proses pembuatan model *classifier* dan juga uji coba tahap *training*. Sedangkan dataset *testing* akan digunakan untuk uji coba namun tidak dicantumkan dalam proses pembuatan model *classifier* seperti dataset *training*. Sampel kode non-*malicious* diambil dari contoh proyek Wordpress versi 4.6.1 dan versi 4.5.0, masing-masing sebanyak 500 file kode PHP. Sebanyak 544 file kode *malicious* diambil dari situs repositori GitHub, dan 101 file kode sumber *malicious* lainnya diambil dari komputer server web Informatika ITS.

C. Hasil Uji Coba

Berikut adalah hasil dari masing-masing skenario uji coba yang telah dijabarkan sebelumnya:

1) Hasil Skenario I: Uji coba tahap training dengan Multinomial Naive Bayes.

Hasil uji coba skenario pertama ditunjukkan oleh Tabel 2. Dari 427 file *malicious*, 362 file terklasifikasi dengan benar namun 65 lainnya terklasifikasi sebagai non-*malicious*. Sedangkan dari 500 file non-*malicious*, 493 file terklasifikasi dengan benar namun 7 lainnya terklasifikasi sebagai *malicious*. Nilai *precision* dari hasil di atas adalah 98,10% (362/(362+7)) dan nilai *recall*-nya adalah 84,77% (362/427).

2) Hasil Skenario II: Uji coba tahap training dengan Decision Tree.

Hasil uji coba skenario kedua ditunjukkan oleh Tabel 3. Semua file *malicious* terklasifikasi dengan benar. Sedangkan dari 500 file non-*malicious*, 464 file terklasifikasi dengan benar tetapi 36 lainnya terklasifikasi sebagai *malicious*. Nilai *precision* dari hasil di atas adalah 92,22% (427/(427+36)) dan nilai *recall*-nya adalah 100% (427/427).

Tabel 2. Hasil klasifikasi dari dataset *training* dengan Multinomial Naive Bayes.

		Prediksi	
		Malicious	**Non-malicious**
Kelas sebenarnya	**Malicious**	362	65
	Non-maicious	7	493

Tabel 3. Hasil klasifikasi dari dataset *training* dengan Decision Tree.

		Prediksi	
		Malicious	**Non-malicious**
Kelas sebenarnya	**Malicious**	427	0
	Non-maicious	36	464

Tabel 4. Hasil klasifikasi dari dataset *testing* dengan Multinomial Naïve Bayes.

		Prediksi	
		Malicious	**Non-malicious**
Kelas sebenarnya	**Malicious**	181	37
	Non-maicious	36	464

Tabel 5. Hasil klasifikasi dari dataset *testing* dengan Decision Tree.

		Prediksi	
		Malicious	**Non-malicious**
Kelas sebenarnya	**Malicious**	212	6
	Non-maicious	81	419

3) Hasil Skenario III: Uji coba tahap testing dengan Multinomial Naive Bayes.

Hasil uji coba skenario ketiga ditunjukkan oleh Tabel 4. Dari 218 file *malicious*, 181 file terklasifikasi dengan benar namun 37 lainnya terklasifikasi sebagai non-*malicious*. Sedangkan dari 500 file non-*malicious*, 464 file terklasifikasi dengan benar tetapi 36 lainnya terklasifikasi sebagai *malicious*. Nilai *precision* dari hasil di atas adalah 83,41% (181/(181+36)) dan nilai *recall*-nya adalah 83,03% (181/218).

4) Hasil Skenario III: Uji coba tahap testing dengan Multinomial Naive Bayes.

Hasil uji coba skenario kelima ditunjukkan oleh Tabel 5. Dari 218 file *malicious*, 212 file terklasifikasi dengan benar namun 6 lainnya terklasifikasi sebagai non-*malicious*. Sedangkan dari 500 file non-*malicious*, 419 file terklasifikasi dengan benar tetapi 81 lainnya terklasifikasi sebagai *malicious*. Nilai *precision* dari hasil di atas adalah 72,35% (212/(212+81)) dan nilai *recall*-nya adalah 97,24% (212/218).

D. Evaluasi Hasil

Dapat dilihat bahwa ternyata hasil klasifikasi menggunakan *classifier Multinomial Naive Bayes* cenderung lebih presisi dibandingkan classifier *Decision Tree*. Namun jumlah file *malware* yang dapat dideteksi oleh *Multinomial Naive Bayes* lebih sedikit daripada *Decision Tree*.

Kedua *classifier* tersebut memiliki performa yang cukup baik untuk mendeteksi kode sumber *malicious*, namun dalam kasus-kasus berbeda, salah satu classifier akan lebih baik daripada *classifier* yang lain. Misalnya apabila pendeteksian *malware* ingin dijalankan secara otomatis pada lingkungan

aplikasi web yang menggunakan kerangka kerja luar atau sistem manajemen konten seperti Wordpress, akan lebih baik apabila menggunakan *classifier Multinomial Naive Bayes*. Dengan *Multinomial Naive Bayes*, kemungkinan file-file non-*malware* krusial terdeteksi sebagai *malware* lebih kecil dibandingkan dengan menggunakan *Decision Tree*. Sedangkan *Decision Tree* dapat digunakan apabila pengguna tidak terlalu mempermasalahkan kesalahan klasifikasi terhadap file-file non-*malicious* dan ingin memaksimalkan kemampuan pendeteksian kode *malicious*.

VI. KESIMPULAN

Berdasarkan perancangan, implementasi, dan pengujian terhadap aplikasi yang dibuat, dapat diambil kesimpulan sebagai berikut:

1. Pendeteksian kode malicious dalam lingkungan aplikasi web berbasis PHP telah dilakukan dengan menerapkan teknik kategorisasi dokumen. Yakni dengan melakukan tokenisasi (mengambil nama variabel, fungsi, kelas, *keyword* eval dan echo, serta konstanta *string*), membuat model *classifier*, dan menggunakan model *classifier* tersebut untuk mengklasifikasi file-file lain.
2. Dari hasil uji coba dengan 218 file *malicious* dan 500 file non-*malicious* didapat bahwa *Multinomial Naive Bayes* dapat mengklasifikasi dengan tingkat *precision* sebesar 83% dan *recall* sebesar 83%. Sedangkan model *Decision Tree* dapat mengklasifikasi dengan tingkat *precision* sebesar 72% dan *recall* hingga 97%. Model *classifier Multinomial Naive Bayes* cenderung lebih presisi daripada *Decision Tree*, namun *Decision Tree* dapat lebih banyak mendeteksi file-file *malicious*.

VII. SARAN

Beberapa saran yang dapat diberikan untuk pengembangan dari topik yang dibahas pada makalah ini adalah sebagai berikut:

1. Tambahkan jenis kode sumber dari aplikasi selain berbasis Wordpress ke dalam dataset *training* dan *testing*. Hal ini dapat dilakukan untuk mengetahui performa teknik kategorisasi dokumen untuk mendeteksi *malware* pada aplikasi-aplikasi web jenis lain.
2. Tambahkan jenis *malware* selain *web shell* ke dalam dataset *training* dan *testing*. Hal ini dapat dilakukan untuk mengetahui performa teknik kategorisasi dokumen untuk mendeteksi *malware-malware* jenis lain.
3. Gunakan metode-metode seleksi fitur dan/atau reduksi dimensionalitas seperti *Principal Component Analysis* sebelum membangun model *classifier*. Karena tiap-tiap jenis token dihitung sebagai satu fitur, maka masing-masing dokumen pada kasus kategorisasi dokumen memiliki banyak fitur. Dari hasil uji coba yang dilakukan diketahui bahwa satu dokumen (satu file kode sumber) memiliki lebih dari 14 ribu fitur. Pada beberapa metode machine learning seperti *Decision Tree*, jumlah fitur yang jauh melebihi jumlah sampel/dokumen akan menyebabkan *overfitting* [6]. Oleh karena itu reduksi dimensionalitas atau seleksi fitur dapat diterapkan untuk mengetahui pengaruhnya terhadap model *classifier* yang dihasilkan.
4. Lakukan uji coba kategorisasi dokumen dengan menggunakan algoritma machine learning selain *Decision Tree* dan *Mutinomial Naive Bayes*. Algoritma-algoritma tersebut dapat diterapkan dan dibandingkan performanya dengan algoritma algoritma yang telah diterapkan pada makalah ini.

DAFTAR PUSTAKA

[1] Clint Feher Asaf Shabtai, Robert Moskovitch and Shlomi Dolev. Detecting unknown malicious code by applying classification techniques on opcode patterns. Security Informatics. doi: 10.1186/2190-8532-1-1.

[2] Fabrizio Sebastiani. Machine learning in automated text categorization. ACM Comput. Surv., 34(1):1–47, March 2002. ISSN 0360-0300. doi: 10.1145/505282.505283.

[3] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. Commun. ACM, 18(11):613–620, November 1975. ISSN 0001-0782. doi: 10.1145/361219.361220.

[4] Harry Zhang. The optimality of Naive Bayes. In Valerie Barr and Zdravko Markov, editors, Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

[6] Decision trees. http://scikit-learn.org/stable/modules/tree.html. Diakses: 2016-12-24.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.