ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

PENDETEKSIAN MALWARE PADA LINGKUNGAN APLIKASI WEB DENGAN KATEGORISASI DOKUMEN

FRANSISKUS GUSTI NGURAH DWIKA SETIAWAN
NRP 5112100013

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

TUGAS AKHIR - KI141502

PENDETEKSIAN MALWARE PADA LINGKUNGAN APLIKASI WEB DENGAN KATEGORISASI DOKUMEN

FRANSISKUS GUSTI NGURAH DWIKA SETIAWAN
NRP 5112100013

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

MALWARE DETECTION IN WEB APPLICATION ENVIRONMENT WITH DOCUMENT CATEGORIZATION

FRANSISKUS GUSTI NGURAH DWIKA SETIAWAN
NRP 5112100013

Supervisor I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Supervisor II
Hudan Studiawan, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2016

(Halaman ini sengaja dikosongkan)

PENDETEKSIAN MALWARE PADA LINGKUNGAN APLIKASI WEB DENGAN KATEGORISASI DOKUMEN

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Fransiskus Gusti Ngurah Dwika Setiawan
NRP: 5112100013

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. .....
NIP: 197708242006041001 (Pembimbing 1)

Hudan Studiawan, S.Kom., M.Kom. .....
NIP: 198705112012121003 (Pembimbing 2)

SURABAYA
Mei 2016

(Halaman ini sengaja dikosongkan)

PENDETEKSIAN MALWARE PADA LINGKUNGAN APLIKASI WEB DENGAN KATEGORISASI DOKUMEN

Nama	:	FRANSISKUS GUSTI NGURAH DWIKA SETIAWAN
NRP	:	5112100013
Jurusan	:	Teknik Informatika FTIf
Pembimbing I	:	Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Pembimbing II	:	Hudan Studiawan, S.Kom., M.Kom.

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 tugas akhir ini, dibuat aplikasi pendeteksian *malware* / 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.

(Halaman ini sengaja dikosongkan)

MALWARE DETECTION IN WEB APPLICATION ENVIRONMENT WITH DOCUMENT CATEGORIZATION

Name	:	FRANSISKUS GUSTI NGURAH DWIKA SETIAWAN
NRP	:	5112100013
Major	:	Informatics FTIf
Supervisor I	:	Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Supervisor II	:	Hudan Studiawan, S.Kom., M.Kom.

Abstract

The number of web applications are growing along with the growth of information technology. As the number of web applications grow, the frequency of attacks against those web applications are increasing too. One example of those attacks is the injection of malware such as web shell to the web application. Such malware can give free access to the application's server for attackers.

In this undergraduate thesis, an application for detecting malware / malicious code using document categorization techniques will be created. The document categorization process consists of preprocessing document, source code tokenization, training of Multinomial Naive Bayes and Decision Tree classifier, and document classification using said classifiers. Tests using 718 PHP source code show that the application successfully detect malware with 72%-83% precision and 83%-97% recall.

Keywords*: web application, malware detection, document categorization.*

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji syukur penulis haturkan kepada Tuhan Yang Maha Esa atas berkat dan rahmatnya sehingga penulis dapat dapat menyelesaikan Tugas Akhir yang berjudul **"Pendeteksian Malware Pada Lingkungan Aplikasi Web dengan Kategorisasi Dokumen"**.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya penulis mendapat sangat banyak bantuan dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar - besarnya kepada:

- Tuhan Yang Maha Esa atas berkat rahmat dan penyertaan-Nya penulis dapat menyelesaikan Tugas Akhir ini.
- Orangtua penulis atas dukungannya sehingga penulis dapat mengerjakan Tugas Akhir ini.
- Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D., Bapak Hudan Studiawan, S.Kom., M.Kom., dan Bapak Baskoro Adi Pratomo, S.Kom., M.Kom. selaku dosen-dosen pembimbing yang telah dengan sabar memberi arahan dan ilmunya terhadap penulis selama pengerjaan Tugas Akhir ini.
- Ibu Dr. Ir. Siti Rochimah, MT., selaku dosen wali penulis, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya kepada penulis.
- Teman-teman penulis, atas semua bantuan dan motivasi yang diberikan hingga Tugas Akhir ini dapat diselesaikan.
- Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya Tugas Akhir ini.

Penulis sadar bahwa Tugas Akhir ini tidak sempurna, oleh karena itu penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Sekian dan Terima Kasih.

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

ABSTRAK **vii**

ABSTRACT **ix**

Kata Pengantar **xi**

1 PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodologi 3
1.7 Sistematika Penulisan 4

2 TINJAUAN PUSTAKA **5**
2.1 Kategorisasi Teks 5
2.2 *Vector Space Model* 5
2.3 *Term Frequency* dan TF-IDF 6
2.4 scikit-learn 7
2.5 *Naive Bayes* 8
2.6 *Decision Tree* 10
2.7 Wordpress 12
2.8 *Web Shell* 12

3 PERANCANGAN **15**
3.1 Deskripsi Umum 15
3.2 Rancangan Modul Praproses 15
3.3 Rancangan Modul *Training* 18
3.4 Rancangan Modul Klasifikasi 22

4 IMPLEMENTASI **27**
4.1 Lingkungan Pembangunan Perangkat Lunak 27
4.1.1 Lingkungan Perangkat Lunak 27
4.1.2 Lingkungan Perangkat Keras 27
4.2 Arsitektur Implementasi Aplikasi 27
4.3 Implementasi Modul Praproses 28
4.4 Implementasi Modul *Training* 32
4.5 Implementasi Modul Klasifikasi 35

5 UJI COBA DAN EVALUASI **39**
5.1 Lingkungan Uji Coba 39
5.2 Data Uji Coba 40
5.3 Skenario Uji Coba 40
5.4 Hasil Uji Coba 42
5.4.1 Hasil Skenario I: Uji coba tahap *training* dengan *Multinomial Naive Bayes* 42
5.4.2 Hasil Skenario II: Uji coba tahap *training* dengan *Decision Tree* 44
5.4.3 Hasil Skenario III: Uji coba tahap *testing* dengan *Multinomial Naive Bayes* 46
5.4.4 Hasil Skenario IV: Uji coba tahap *testing* dengan *Decision Tree* 47
5.5 Evaluasi Hasil 47

6 PENUTUP **51**
6.1 Kesimpulan 51
6.2 Saran 52

DAFTAR PUSTAKA **53**

LAMPIRAN **55**
A.1 Kode sumber *tokenizer* 55
A.2 Kode sumber modul praproses 59
A.3 Kode sumber modul *training* 60

A.4 Kode sumber modul klasifikasi 66

BIODATA PENULIS **71**

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

2.1 Contoh nilai *term frequency* untuk masing-masing pasangan *term* dan dokumen 6

5.1 Skenario-skenario uji coba. 40
5.2 Sepuluh token dengan nilai feature_log_prob tertinggi untuk kelas *malicious*. 43
5.3 Sepuluh token dengan nilai feature_log_prob tertinggi untuk kelas non-*malicious*. 44
5.4 Hasil prediksi dataset *training* dengan *classifier Multinomial Naive Bayes*. 44
5.5 Hasil prediksi dataset *training* dengan *classifier Decision Tree*. 46
5.6 Hasil prediksi dataset *testing* dengan *classifier Multinomial Naive Bayes*. 47
5.7 Hasil prediksi dataset *testing* dengan *classifier Decision Tree*. 47
5.8 Nilai *precision* dan *recall* dari masing-masing *classifier*. 48

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

3.1 Diagram relasi antar modul. 16
3.2 Diagram alir tahap praproses. 17
3.3 Diagram alir tahap *training*. 18
3.4 Diagram alir tahap klasifikasi. 23
3.5 Diagram alir proses klasifikasi oleh objek *MultinomialNB*. 24
3.6 Diagram alir proses klasifikasi oleh objek *DecisionTreeClassifier*. 25

4.1 Diagram arsitektur implementasi aplikasi. 28

5.1 Diagram arsitektur lingkungan uji coba. 39
5.2 Ilustrasi model *classifier Decision Tree* hasil proses *training* hingga kedalaman 5. 45

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Bab ini membahas garis besar penyusunan tugas akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan tugas akhir dan sistematika penulisan.

1.1 Latar Belakang

Seiring berkembangnya teknologi informasi, permintaan akan aplikasi berbasis web semakin meningkat. Jumlah layanan *web hosting* pun bertambah selaras dengan permintaan aplikasi web. Layanan *web hosting* yang ditawarkan pun bervariasi, mulai dari layanan bebas biaya dengan fitur terbatas hingga layanan berbayar dengan fitur lanjut dan dukungan profesional. Layanan *web hosting* bebas biaya dengan fitur terbatas hingga layanan bebas biaya biasanya lebih diminati oleh konsumen. Namun, seringkali konsumen-konsumen tersebut melupakan beberapa hal penting seperti aspek keamanan. Hal ini membuka peluang bagi orang-orang yang tidak bertanggung jawab untuk menyerang aplikasi web mereka. Salah satu jenis serangan yang umum dilakukan adalah penyisipan *malware*. Malware tersebut biasanya berupa kode sumber berbahaya (*malicious*) yang disisipkan dalam sebuah file, seperti file gambar, dll. Efek dari malware tersebut bervariasi, mulai dari pengubahan tampilan halaman web (*deface*), hingga pencurian data-data sensitif. Untuk itu dibutuhkan suatu cara untuk mendeteksi *malware*-*malware* tersebut dan menghapus / menonaktifkannya. Salah satu metode pendeteksian *malware* yang pernah dilakukan adalah pendeteksian kode sumber *malicious* dengan 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* ada-

lah bagian dari instruksi bahasa mesin yang mendefinisikan operasi apa yang harus dilakukan. Dalam riset tersebut, dilakukan beberapa eksperimen untuk menentukan nilai-nilai parameter paling baik untuk pendeteksian malware dari pola *OpCode*. Parameter-parameter tersebut adalah jenis *classifier*, metode tokenisasi, metode pembobotan term, metode reduksi fitur, dan persentase jumlah *malware* dan non-*malware* dalam data belajar. Dari riset tersebut, diketahui kategorisasi dokumen dengan menggunakan classifier *Boosted Decision Tree*, berhasil mendeteksi *malware* dengan tingkat akurasi mencapai 94,5%. Hal ini menunjukkan bahwa teknik kategorisasi dokumen dapat diterapkan untuk melakukan pendeteksian *malware* dalam file-file biner.

Oleh karena itu, dalam tugas akhir ini penulis mengajukan implementasi teknik kategorisasi dokumen untuk mendeteksi *malware* dalam aplikasi berbasis web. Karena struktur dan tata bahasa kode sumber dalam aplikasi *web* berbeda dengan *OpCode*, maka akan ada beberapa parameter yang harus disesuaikan. Nilai-nilai parameter tersebut akan ditentukan melalui eksperimen.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana penerapan teknik kategorisasi dokumen untuk mendeteksi kode malicious dalam aplikasi web?
2. Bagaimana metode *Naive Bayes* dan *Decision Tree* dapat mengklasifikasi kode sumber aplikasi web?
3. Bagaimana metode tokenisasi dokumen untuk kasus pendeteksian malware dalam apilkasi web?

1.3 Batasan Masalah

Permasalahan yang dibahas pada tugas akhir ini memiliki beberapa batasan, yaitu:

1. Kode sumber yang dapat diklasifikasi adalah kode sumber dalam bahasa PHP.

2. Jenis *malware* yang akan diklasifikasi adalah *Web Shell*
3. Jenis aplikasi web yang digunakan untuk uji coba adalah aplikasi web berbasis *Wordpress*
4. Aplikasi yang dibuat menggunakan antar muka *command line*.

1.4 Tujuan

Tujuan pengerjaan tugas akhir ini adalah membuat aplikasi yang dapat mendeteksi *malware* dari file-file dalam sebuah aplikasi web dengan menggunakan teknik kategorisasi dokumen.

1.5 Manfaat

Manfaat pembuatan tugas akhir ini adalah menyediakan suatu perangkat lunak yang dapat membantu mendeteksi file-file malware dalam aplikasi web secara efektif, sehingga dapat meningkatkan keamanan aplikasi web.

1.6 Metodologi

Adapun langkah - langkah yang ditempuh dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1. Studi literatur
   Pada tahap ini dilakukan studi literatur mengenai metode yang digunakan. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal dan dokumentasi dari internet.
2. Pengumpulan data
   Pada tahap ini, data-data yang diperlukan untuk aplikasi dan pengujian aplikasi akan dikumpulkan. Data-data yang dikumpulkan berupa *file-file* kode sumber PHP *malicious* dan non-*malicious*.
3. Perancangan dan implementasi
   Dalam tahap ini dilakukan perancangan terhadap struktur aplikasi yang akan dibangun. Hasil perancangan tersebut akan digunakan dalam proses implementasi aplikasi.
4. Uji coba dan evaluasi

Uji coba dan evaluasi dilakukan setelah aplikasi selesai diimplementasi dan data uji coba dikumpulkan. Pengujian dilakukan untuk mengukur seberapa efektif dan akurat aplikasi dalam mengklasifikasi kode sumber aplikasi web.

5. Penyusunan buku tugas akhir
   Pada tahapan ini buku tugas akhir disusun sebagai dokumentasi dari pelaksanaan tugas akhir.

1.7 Sistematika Penulisan

Buku tugas akhir ini disusun degnan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

BAB II. TINJAUAN PUSTAKA

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan tugas akhir ini.

BAB III. PERANCANGAN

Bab ini berisi perancangan alur kerja aplikasi dan perancangan modul-modul yang nantinya akan diimplementasikan dan dilakukan uji coba.

BAB IV. IMPLEMENTASI

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Dalam bab ini akan dicantumkan pseudocode untuk masing-masing modul aplikasi.

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan tahap pengujian sistem dan pengujian performa aplikasi beserta hasilnya.

BAB VI. PENUTUP

Bab ini menyampaikan kesimpulan dari hasil uji coba yang dilakukan terhadap rumusan masalah yang ada dan saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan tugas akhir ini.

2.1 Kategorisasi Teks

Kategorisasi teks adalah pekerjaan pemberian label kategori kepada sebuah dokumen teks [2]. Label kategori tersebut diambil dari kumpulan kategori-kategori yang telah ditentukan sebelumnya. Proses ini dapat dilakukan secara manual oleh manusia atau secara otomatis oleh komputer dengan *machine learning*. Secara formal kategorisasi teks dapat didefinisikan sebagai kegiatan pemberian nilai *Boolean* T atau F (*true* atau *false*) kepada setiap pasangan dokumen-kategori $\langle d_i, c_j \rangle \in D \times C$, di mana D adalah himpunan dokumen dan C adalah himpunan kategori. Apabila bernilai T, maka dapat diartikan bahwa dokumen d_i termasuk dalam kategori c_j. Sebaliknya apabila bernilai F, dokumen d_i tidak termasuk dalam kategori c_j. Untuk dapat melakukan kategorisasi teks secara otomatis, perlu diketahui fungsi target $\phi' : D \times C \rightarrow \{T, F\}$ yang mendeskripsikan bagaimana dokumen seharusnya diklasifikasi. Fungsi target ϕ' tidak dapat diketahui secara langsung, oleh karena itu dibutuhkan *machine learning* untuk mencari pendekatan fungsi tersebut. Algoritma-algoritma *machine learning* seperti *Naive Bayes* dan *Decision Tree* akan diterapkan untuk mencari fungsi $\phi : D \times C \rightarrow \{T, F\}$ yang mendekati fungsi target ϕ'. Dalam kasus kategorisasi malware, hanya ada satu kategori yakni apakah dokumen termasuk malware atau bukan. Oleh karena itu fungsi target dapat disederhanakan menjadi $\phi : D \rightarrow \{0, 1\}$, begitu juga dengan fungsi *classifier*.

2.2 *Vector Space Model*

Vector Space Model adalah model matematis untuk merepresentasikan dokumen teks. Sebuah dokumen d yang di dalamnya ter-

Tabel 2.1: Contoh nilai *term frequency* untuk masing-masing pasangan *term* dan dokumen

Term \ Dokumen	Antony and Cleopatra	Julius Caesar	Hamlet
"antony"	157	73	0
"brutus"	4	157	1
"caesar"	232	227	2
"cleopatra"	57	0	0
"mercy"	2	0	5
"worser"	2	0	1

.

dapat n kata / *term* unik $t_1, t_2, ..., t_n$ memiliki representasi vektor $\overrightarrow{d} = (w_1, w_2, ..., w_n)$, dengan w_i adalah bobot atau nilai dari term t_i untuk dokumen d. Representasi vektor ini biasa digunakan dalam sistem temu kembali informasi (information retrieval system), 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, dan biasanya ditentukan dengan eksperimen.

2.3 *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.

Nilai-nilai pada tabel 2.1 adalah jumlah kemunculan term (TF) pada dokumen yang bersangkutan. Apabila suatu term tidak muncul sama sekali di suatu dokumen, maka nilai TF untuk pasangan term-dokumen tersebut adalah 0. Semakin panjang suatu dokumen, maka nilai TF masing-masing term-nya cenderung semakin besar. Untuk mengurangi efek ini, seringkali digunakan bentuk lain dari

pembobotan TF, misalnya seperti persamaan 2.1

$$TF'_{t,d} = \begin{cases} 1 + log_{10}\, TF_{t,d} & TF_{t,d} > 0 \\ 0 & TF_{t,d} = 0 \end{cases} \quad (2.1)$$

TF_t, d adalah nilai TF untuk *term* t dan dokumen d.

Dengan menggunakan logaritma, maka pertumbuhan nilai w_t, d seiring dengan TF_t, d tidak akan terlalu pesat. Hasilnya 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 ini mempertimbangkan seberapa unik suatu *term* dalam kumpulan dokumen. Diasumsikan 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 Indonsia, *term* (kata) "di", "dan", "ke", "yang", "atau", dan "dan" akan memiliki nilai bobot yang relatif lebih rendah. Dari situ muncul persamaan pembobotan TF-IDF seperti yang ditunjukkan oleh persamaan 2.2

$$TFIDF'_{t,d} = \begin{cases} 1 + log_{10}\, TF_{t,d} \times N/df_t & TF_{t,d} > 0 \\ 0 & TF_{t,d} = 0 \end{cases} \quad (2.2)$$

Dengan:

- N adalah jumlah semua dokumen
- df_t adalah *document frequency* dari *term* t, yakni jumlah dokumen yang mengandung *termt*

2.4 scikit-learn

scikit-learn adalah pustaka *machine learning* yang dibuat untuk bahasa Python [4]. Pustaka ini menyediakan kelas-kelas untuk proses klasifikasi, regresi, clustering, dan juga beberapa kelas-kelas

lain untuk melakukan pembacaan data, praproses data, dan lain-lain. *scikit-learn* menyediakan beberapa kelas yang berguna untuk proses kategorisasi dokumen, seperti *CountVectorizer* dan *TfidfTransformer*. Kelas CountVectorizer berfungsi untuk mengubah dokumen menjadi representasi vektornya dengan pembobotan TF. Sedangkan TfidfTransformer berfungsi untuk mentransformasi vektor dengan pembobotan TF menjadi vektor dengan pembobotan TF-IDF. Untuk membantu membangun classifier, pustaka ini menyediakan kelas-kelas seperti MultinomialNB (implementasi classifier *Multinomial Naive Bayes*), DecisionTreeClassifier (implementasi classifier Decision Tree), dan lain-lain.

2.5 *Naive Bayes*

Naive Bayes adalah 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) [5]. Teorema Bayes menyatakan bahwa peluang dari sebuah sampel $d = (x_1, x_2, ..., x_n)$ termasuk dalam sebuah kelas c adalah:

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)} \tag{2.3}$$

Dengan mengasumsikan bahwa masing-masing fitur pada d tidak bergantung satu sama lain, maka persamaan di atas menjadi:

$$\begin{aligned} p(c|d) &= \frac{p(c)}{p(d)} \prod_{i=1}^{n} p(x_i|c) \\ &\propto p(c) \prod_{i=1}^{n} p(x_i|c) \end{aligned} \tag{2.4}$$

Data d termasuk dalam kelas dengan nilai $p(c|d)$ tertinggi untuk $c \in C$ di mana C adalah himpunan semua kelas yang mungkin untuk data d.

Salah satu bentuk dari algoritma *Naive Bayes* untuk mengklasifikasi dokumen adalah *Multinomial Naive Bayes* [6]. Dalam penerapan *Multinomial Naive Bayes* untuk klasifikasi dokumen, nilai $x_i, i = 1...n$ merupakan nilai numerik, maka dari itu $p(c|d)$ didefinisikan seperti pada persamaan 2.5.

$$p(c|d) \propto p(c) \prod_{i=1}^{n} p^{x_i}(x_i|c) \tag{2.5}$$

Nilai $p(x|c)$ merupakan frekuensi relatif term x dari dokumen-dokumen dalam kelas c. Nilai tersebut didefinisikan seperti pada persamaan 2.6.

$$p(x|c) = \frac{T_{x,c}}{\sum_{x' \in X} T_{x',c}} \tag{2.6}$$

di mana:

- $T_{x,c}$ adalah jumlah nilai TF atau tf-idf dari *term* x dari semua dokumen dalam kelas c,
- X adalah himpunan *term*

Terkadang dalam suatu kelas tidak terdapat sama sekali term x. Hal ini menyebabkan nilai T_x menjadi nol. Apabila nilai $T_x = 0$ dibiarkan, maka nilai $p(x|c)$ pasti nol. Untuk mencegah hal ini, biasanya nilai T_x ditambah satu, sehingga persamaan 2.6 akan menjadi seperti persamaan 2.7.

$$p(x|c) = \frac{T_{x,c} + 1}{\sum_{x' \in X} T_{x',c} + 1} \tag{2.7}$$

Untuk mencegah *underflow*, biasanya nilai yang dihitung adalah bentuk logaritma dari $p(c|d)$ seperti dalam implementasi *Multinomial Naive Bayes* pada pustaka *scikit-learn*. Oleh karena itu

persamaan 2.5 menjadi seperti persamaan 2.8.

$$\log p(c|d) \propto \log p(c) \sum_{i=1}^{n} x_i \log p(x_i|c) \tag{2.8}$$

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

2.6 *Decision Tree*

Decision Tree adalah salah satu jenis algoritma *machine learning* yang lazim digunakan untuk klasifikasi maupun regresi [7]. Terdapat beberapa jenis algoritma *Decision Tree*. Beberapa di antaranya adalah: ID3, C4.5, dan CART. Algoritma *Decision Tree* yang digunakan oleh pustaka *scikit-learn* adalah CART (*Classification and Regression Tree*). Algoritma CART mendukung jenis data numerik sebagai fitur, dan juga mendukung jenis data variabel target numerik seperti pada kasus regresi.

Tujuan dari algoritma *Decision Tree* secara umum adalah memprediksi nilai dari variabel target y dari sampel x yang memiliki n fitur $x_1, x_2, ..., x_n$. Hal ini dilakukan dengan membuat sebuah *tree*. Pada algoritma CART, jenis *tree* yang dibuat adalah *binary tree*. Untuk dapat membangun sebuah model *Decision Tree*, diperlukan sebuah kumpulan data *training* yang terdiri dari kumpulan sampel-sampel yang nilai variabel targetnya sudah diketahui. Jadi untuk tiap sampel x^i pada dataset *training*, nilai variabel target y^i sudah diketahui.

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 x_j. Apabila sampel x^i memiliki fitur x_j^i yang bernilai kurang dari sebuah nilai s_j, maka sampel tersebut masuk ke *node* cabang kiri dari *node* sekarang. Apabila x_j^i bernilai lebih dari atau sama

dengan s_j maka sampel tersebut masuk ke dalam *node* cabang kanan dari *node* sekarang. Proses tersebut dilakukan secara rekursif hingga terbentuk *node-node 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 dalam 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 (variabel target y adalah data kategorikal), nilai *gini impurity* untuk suatu node n dapat didefinisikan seperti pada persamaan 2.9.

$$I(n) = \sum_{y \in Y} f_{n,y}(1 - f_{n,y}) \tag{2.9}$$

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 persamaan 2.10.

$$\Delta I(n) = I(n) - I(n_l) - I(n_r) \tag{2.10}$$

dengan:

- $\Delta I(n)$ adalah perubahan nilai *gini impurity*,
- $I(n_l)$ adalah nilai *gini impurity* untuk *node* cabang kiri,
- $I(n_r)$ adalah nilai *gini impurity* untuk *node* cabang kanan

Nilai $\Delta I(n)$ dihitung untuk tiap-tiap fitur yang ada. Fitur yang dapat menghasilkan pemecahan dataset dengan nilai $\Delta I(n)$ terbesar

akan digunakan untuk memecah dataset pada *node* n.

Data baru dapat diklasifikasikan 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 nilai kelas dominan dari *node leaf* yang dicapai.

2.7 Wordpress

Wordpress adalah sebuah sistem manajemen konten (CMS) sumber terbuka berbasis PHP dan MySQL [8]. Pada tahun 2016, Wordpress telah digunakan untuk membangun 27% situs web di internet [9]. Beberapa fitur yang disediakan Wordpress adalah sistem *template* web, kumpulan tema, dan arsitektur *plugin*. Pengguna Wordpress dapat dengan bebas memasang tema, baik itu tema bebas (gratis) maupun premium (berbayar). Dengan tema, pengguna dapat mengganti tampilan situs web mereka. Selain mengganti tampilan, pengguna dapat menambah fungsionalitas dari situs mereka dengan menggunakan *plugin*. Kumpulan tema dan *plugin* yang dapat digunakan pada *Wordpress* biasanya dikembangkan oleh pengembang pihak ketiga. Tema-tema dan *plugin-plugin* tersebut ditinjau ulang oleh pihak *Wordpress* sebelum ditambahkan ke repositori tema dan *plugin Wordpress* [10]. Hal ini dilakukan untuk menjaga kualitas dari tema dan *plugin* yang diunggah ke repositori *Wordpress*. Namun sayangnya, meskipun telah melalui proses peninjauan ulang, beberapa tema dan *plugin* masih memiliki celah keamanan yang dapat mengancam situs-situs penggunanya. Berdasarkan hasil pemindaian yang dilakukan oleh Checkmarx pada Juni 2013, dari 20% dari 50 *plugin* terpopuler *Wordpress* memiliki celah keamanan terhadap serangan SQL *injection*, *cross-site scripting*, *cross-site request forgery*, dan *path traversal* [11].

2.8 *Web Shell*

Web Shell adalah skrip yang diunggah ke sebuah server web untuk memberikan akses *shell* kepada pengunggah atau pengguna

lainnya secara jarak jauh [12]. Skrip *web shell* dapat ditulis dalam berbagai bahasa yang didukung oleh server web. Bahasa-bahasa yang paling banyak digunakan untuk menulis *web shell* meliputi PHP, ASP, Perl, Ruby, Python, dan skrip *shell* Unix.

Penyerang dapat memanfaatkan celah-celah keamanan yang terdapat pada aplikasi web ataupun sistem manajemen konten (CMS) untuk mengunggah sebuah skrip *web shell* ke sebuah server web. Metode-metode serangan yang dapat digunakan untuk mengunggah skrip *web shell* meliputi: *cross-site scripting*, SQL *injection*, *remote file inclusion*, dan celah-celah spesifik yang terdapat pada *plugin-plugin* CMS. Setelah skrip *web shell* berhasil diunggah, penyerang dapat memanfaatkan teknik eksploitasi lain untuk mendapatkan hak akses yang lebih tinggi seperti hak akses admin. Dengan menggunakan *web shell*, seorang penyerang dapat menjalankan berbagai jenis perintah pada komputer server yang telah terinfeksi.

Ada beberapa jenis *web shell* yang tersebar di internet. Beberapa di antaranya adalah China Chopper, WSO, C99, dan B374K [12]. Masing-masing jenis *web shell* memiliki fitur yang berbeda-beda. Contohnya China Chopper menyediakan fitur untuk menebak kata sandi secara *brute force*, C99 dapat menampilkan metode-metode pengamanan yang diterapkan pada server, dan B374K dapat menampilkan proses-proses yang berjalan pada server.

(Halaman ini sengaja dikosongkan)

BAB III

PERANCANGAN

Bab ini secara khusus akan menjelaskan rancangan aplikasi yang dibuat dalam Tugas Akhir ini. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum hingga perancangan masing-masing komponen.

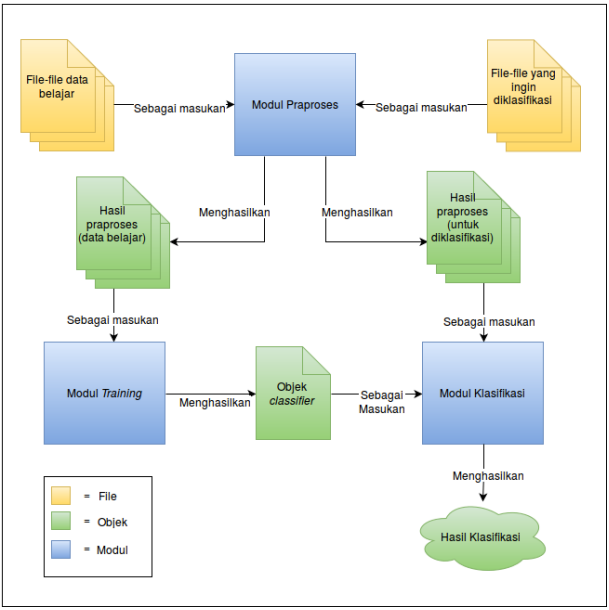
3.1 Deskripsi Umum

Untuk dapat melakukan pendeteksian kode berbahaya / *malicious* pada komputer server *web*, perlu dibangun suatu aplikasi yang dapat menjalankan dua fungsi utama, yakni melakukan *training* / pembelajaran dari data belajar, dan melakukan klasifikasi terhadap suatu file selain dari data belajar. Aplikasi pendeteksi kode *malicious* yang akan dibangun akan terdiri dari tiga modul terpisah: modul praproses, modul *training*, dan modul klasifikasi. Ketiga modul tersebut menangani tiga proses yang berbeda, namun saling berkaitan. Modul praproses akan menghasilkan luaran yang akan digunakan oleh modul *training*. Begitu juga dengan modul *training*, modul *training* akan menghasilkan luaran yang akan digunakan oleh modul klasifikasi. Diagram pada Gambar 3.1 menggambarkan hubungan antar modul dalam aplikasi.

3.2 Rancangan Modul Praproses

Aplikasi ini menerima file-file dalam berbagai format sebagai data belajar/*training* atau untuk diklasifikasi. Sebelum dapat digunakan sebagai data belajar / diklasifikasi, file-file tersebut harus diproses terlebih dahulu. Dalam tahap praproses, isi dari sebuah file akan ditransformasi ke dalam format yang sesuai untuk tahap *training* dan klasifikasi. Modul praproses adalah modul yang bertujuan untuk melakukan tahap praproses tersebut.

Modul praproses mengambil masukan berupa sebuah file dalam format apapun, lalu mengambil komponen kode sumber PHP dalam file tersebut apabila memang ditemukan. Kemudian dari kode



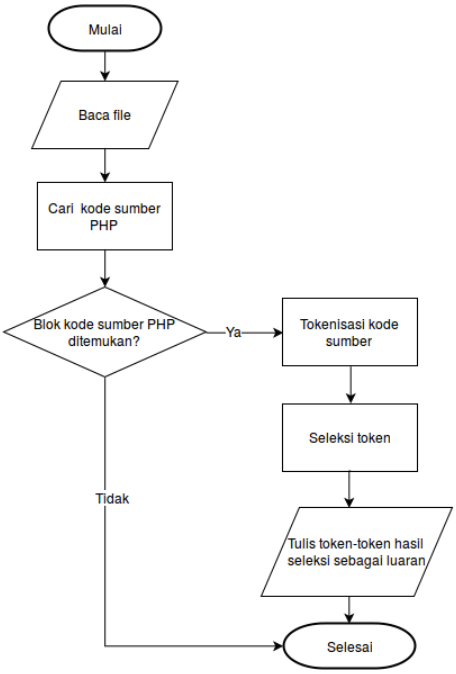
Gambar 3.1: Diagram relasi antar modul.

sumber PHP yang telah terbaca tersebut, akan dilakukan tokenisasi kode sumber. Dalam proses tokenisasi, kode sumber akan dipecah menjadi kumpulan-kumpulan *token*. Jenis-jenis *token* yang diambil adalah nama variabel, nama *class*, nama fungsi, dan *string literal*. Untuk jenis *token string literal* dilakukan pengelompokan seperti berikut:

1. *Token-token* berupa *string* panjang (lebih dari nilai *threshold* panjang tertentu) akan diubah menjadi *string* ”LONG_NORMAL_STRING”.
2. *Token-token* dalam format *base64* akan diubah menjadi *string* ”LONG_BASE64_STRING”.
3. *Token-token* yang memiliki representasi karakter dalam bentuk hexadesimal (seperti ”\x8F”) di dalamnya akan diubah

menjadi *string* ”STRING_WITH_HEX_LITERAL”.

Hasil dari proses tokenisasi merupakan luaran dari modul praproses. Hasil ini akan digunakan oleh modul *training* atau modul klasifikasi untuk tahap selanjutnya. Diagram alir untuk tahap praproses dalam modul praproses dapat dilihat pada Gambar 3.2. Modul

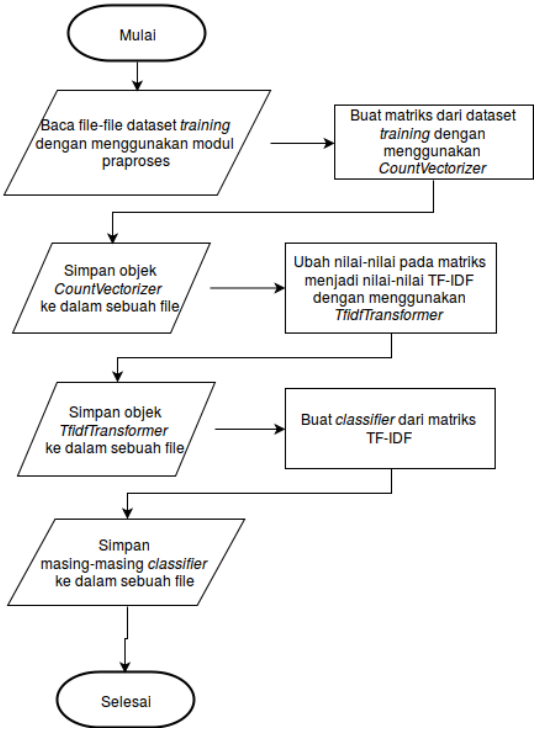


Gambar 3.2: Diagram alir tahap praproses.

praproses tidak dijalankan secara langsung oleh pengguna aplikasi. Modul ini akan digunakan oleh modul *training* dan modul klasifikasi ketika ingin membaca suatu file masukan. Untuk itu dalam modul praproses akan didefinisikan sebuah fungsi yang dapat dipanggil oleh modul-modul lainnya untuk melakukan praproses.

3.3 Rancangan Modul *Training*

Modul *Training* adalah modul yang menangani proses *training* / pembelajaran dari masukan data belajar. Hasil dari proses *training* adalah objek *classifier* yang nantinya dapat digunakan untuk mengklasifikasi kode sumber baru. Diagram alir untuk proses pada modul *training* dapat dilihat pada Gambar 3.3.



Gambar 3.3: Diagram alir tahap *training*.

Sebelum proses training dimulai, modul ini mengambil masukan berupa data belajar. Data belajar dapat dibaca dari direktori tertentu, atau dari kumpulan file yang telah digabungkan dan terkompresi dalam format *.zip. Masing-masing file dalam data belajar ak-

an dipraproses dengan menggunakan fungsi dari modul praproses. Hasil praproses dari masing-masing file akan disimpan ke dalam sebuah objek *list*.

Agar dapat digunakan dalam proses *training*, masing-masing entri dalam objek *list* tersebut harus diubah ke dalam bentuk vektor. Untuk melakukan pengubahan dokumen menjadi vektor, modul ini akan menggunakan objek *CountVectorizer* dari pustaka *scikit-learn*.

Mula-mula objek *CountVectorizer* tersebut akan membangun kamus yang berisi semua *n-gram* unik dari seluruh dokumen. Kemudian objek ini akan menghasilkan matriks M berukuran $m \times n$, dengan m adalah jumlah dokumen dan n adalah jumlah *token* unik dari seluruh dokumen. Masing-masing baris pada M merepresentasikan sebuah dokumen, dan masing-masing kolomnya merepresentasikan sebuah *n-gram*. Pada matriks M, elemen M_i, j berisi jumlah kemunculan *n-gram* ke-j pada dokumen ke-i. Nilai-nilai tersebut kemudian akan diubah menjadi nilai-nilai *TF-IDF* dengan menggunakan objek *TfidfTransformer* dari pustaka *scikit-learn*.

Dari dataset *training* tersebut, modul ini akan membangun objek-objek *classifier* dengan menggunakan teknik *machine learning Multinomial Naive Bayes* dan *Decision Tree*. Proses pembangunan objek-objek *classifier* tersebut dilakukan dengan menggunakan kelas-kelas dari pustaka *scikit-learn*, yakni *MultinomialNB* dan *DecisionTreeClassifier*. Kedua kelas tersebut dapat menerima matriks M beserta sebuah *list* yang berisi label dari masing-masing dokumen sebagai data masukan. Secara garis besar, algoritma pembuatan model *Multinomial Naive Bayes* dan *Decision Tree* yang dilakukan oleh kelas-kelas dalam pustaka *scikit-learn* ditunjukkan oleh Algoritma 3.1 dan 3.2.

Proses pembuatan model *Multinomial Naive Bayes* menerima M sebagai data belajar. Mula-mula dibuat sebuah objek *classifier* baru. Lalu untuk masing-masing kelas c (dalam hal ini *malicious* dan non-*malicious*) pada dataset, akan dihitung nilai $p(c)$, yakni probabilitas munculnya kelas c dalam dataset. Nilai-nilai tersebut ak-

Algoritma 3.1: Pembuatan model *Multinomial Naive Bayes*

```
Function train_multinomial_naive_bayes(M) begin
    T ← semua token pada M;
    C ← semua kelas pada M;
    MNB ← objek classifier Multinomial Naive Bayes
     baru;
    MNB.class_log_prob_ ← list baru;
    MNB.feature_log_prob_ ← list baru;
    for i = 0 to |C| do
        MNB.class_log_prob_[i] ← p(C[i]) for j = 0 to
         |T| do
            MNB.feature_log_prob_[i][j] ←
             p(T[j]|C[i])
        end
    end
    return MNB;
end
```

an disimpan dalam sebuah properti bernama *class_log_prob_* pada objek *classifier* yang dibuat sebelumnya. Setelah itu, untuk masing-masing pasangan *token* t dan kelas c pada dataset, akan dihitung nilai $p(t|c)$. Perhitungan nilai $p(t|c)$ dilakukan dengan menggunakan persamaan 2.7. Nilai $p(t|c)$ adalah frekuensi relatif token t dari dokumen-dokumen yang termasuk dalam kelas c. Masing-masing nilai $p(t|c)$ yang terhitung akan disimpan dalam properti *feature_log_prob_* dalam objek *classifier* yang telah dibuat.

Proses pembuatan model *Decision Tree* juga menerima M sebagai data belajar. Mula-mula sebuah *node* awal dibuat berisi seluruh data dari M. Kemudian nilai perubahan *gini impurity* akan dihitung untuk masing-masing *token* dalam dataset. Nilai perubahan *gini impurity* dihitung menurut persamaan 2.10. *Token* dengan perubahan *gini impurity* tertinggi akan digunakan untuk memecah

Algoritma 3.2: Pembuatan model *Decision Tree*

Function `train_decision_tree(`$M, threshold$`)` **begin**
    $N \leftarrow$ node baru;
    $N.dataset \leftarrow$ sampel-sampel dari M;
    $Tree \leftarrow make\_tree(N, threshold)$;
    return $Tree$;
end

Function `make_tree(`$N, threshold$`)` **begin**
    $D \leftarrow N.dataset$;
    $F \leftarrow$ fitur-fitur dari dataset D;
    if $|D| < threshold$ atau *hanya ada satu kelas pada* D **then**
        return;
    end
    $max\_delta\_I \leftarrow 0$;
    $D_L, D_R \leftarrow$ dataset kosong;
    for f *in* F **do**
        `// pecah dataset` D `berdasarkan fitur` f `menjadi dua dataset baru`
        $D'_L, D'_R \leftarrow split(D, f)$;
        `// hitung perubahan gini impurity`
        $delta\_I \leftarrow gini\_impurity(D)$ - $gini\_impurity(D_L)$ - $gini\_impurity(D_R)$;
        if $delta\_I > max\_delta\_I$ **then**
            $max\_delta\_I \leftarrow delta\_I$;
            $D_L, D_R \leftarrow D'_L, D'_R$
        end
    end
    $N.kiri \leftarrow$ node baru berisi D_L;
    $N.kanan \leftarrow$ node baru berisi D_R;
    $make\_tree(N.kiri, threshold)$;
    $make\_tree(N.kanan, threshold)$;
end

dataset menjadi dua dataset pecahan eksklusif. Dua *node* baru akan dibuat sebagai cabang dari *node* awal. Masing-masing *node* akan satu pecahan dataset. Proses penghitungan *gini impurity* dan pemecahan dataset ini dilakukan secara rekursif hingga jumlah data pada dataset kurang dari nilai *threshold*, atau hanya ada satu kelas dalam dataset.

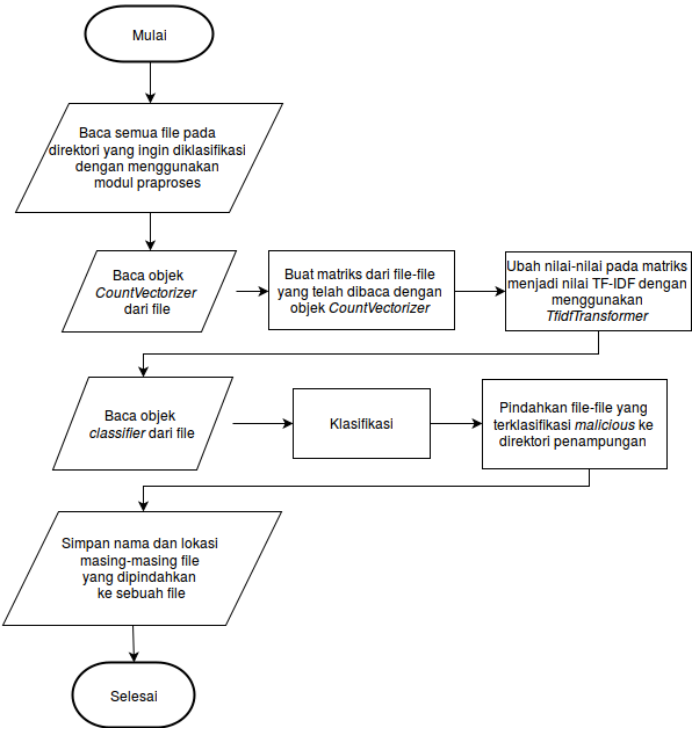
Masing-masing objek *classifier* yang dihasilkan akan disimpan ke dalam sebuah file yang dapat diakses oleh modul klasifikasi. Selain objek-objek *classifier*, objek-objek yang digunakan untuk mengubah dokumen ke dalam bentuk matriks (*CountVectorizer*, *TfidfTransformer*, dan objek-objek *feature_selection*) juga masing-masing disimpan ke dalam sebuah file.

3.4 Rancangan Modul Klasifikasi

Modul klasifikasi bertujuan untuk melakukan pengecekan terhadap file-file dalam sebuah direktori apakah mengandung kode sumber PHP *malicious* atau tidak. Modul ini menerima masukan berupa direktori yang berisi file-file yang ingin diklasifikasi dan juga objek-objek hasil dari tahap *training*. Objek-objek tersebut meliputi: *CountVectorizer*, *TfidfTransformer*, objek-objek *feature_selection*, dan objek *MultinomialNB* atau *DecisionTreeClassifier*.

Sebelum menjalankan proses klasifikasi, pengguna harus mendefinisikan opsi-opsi berikut: direktori dari file-file yang ingin diklasifikasi, direktori tempat file-file yang teridentifikasi *malicious* akan ditampung, objek seleksi fitur yang akan digunakan, dan objek *classifier* yang akan digunakan. Semua itu ditulis ke dalam sebuah file konfigurasi yang nantinya akan dibaca oleh modul klasifikasi.

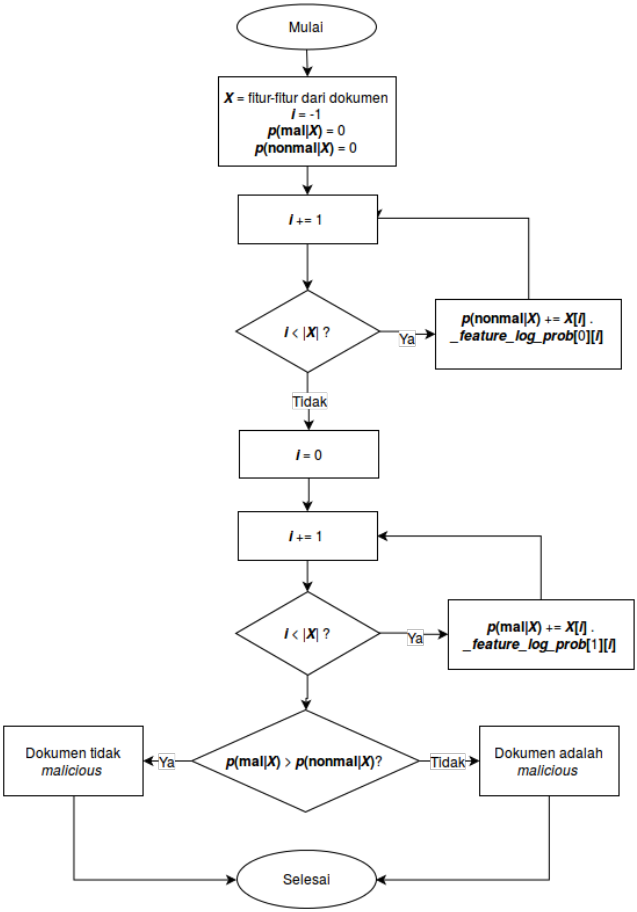
Diagram alir untuk proses dalam modul klasifikasi dapat dilihat pada Gambar 3.4. Proses klasifikasi dimulai dengan membaca file-file dari direktori yang telah ditentukan. Pembacaan file dilakukan dengan menggunakan fungsi dari modul praproses. Hasil praproses dari masing-masing file akan digabungkan ke dalam sebuah objek *list*. Dari objek *list* tersebut akan dibentuk matriks dengan menggunakan objek *CountVectorizer*. Setelah itu akan dilakukan konversi



Gambar 3.4: Diagram alir tahap klasifikasi.

nilai-nilai pada matriks menjadi nilai *TF-IDF* dan seleksi fitur dengan menggunakan objek *TfidfTransformer* dan *feature_selection*. Kemudian dokumen-dokumen yang direpresentasikan dalam bentuk matriks tersebut akan diklasifikasi dengan menggunakan objek *classifier* yang telah ditentukan. Terdapat dua opsi objek *classifier* yang dapat digunakan, objek *MultinomialNB* yang mengimplementasikan *classifier Multinomial Naive Bayes* dan objek *DecisionTreeClassifier* yang mengimplementasikan *classifier Decision Tree*.

Secara garis besar, algoritma pengklasifikasian dokumen yang dilakukan oleh objek model *MultinomialNB* dan *DecisionTreeClas-*

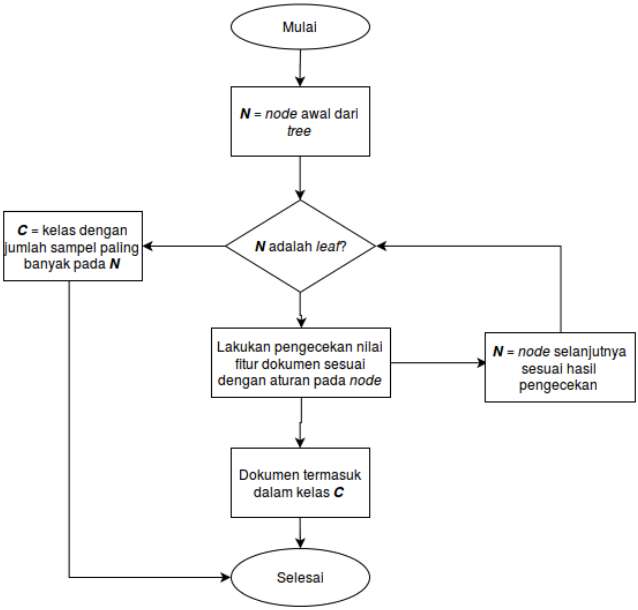


Gambar 3.5: Diagram alir proses klasifikasi oleh objek *MultinomialNB*.

sifier ditunjukkan oleh diagram alir pada Gambar 3.5 dan 3.6 Objek *MultinomialNB* mengklasifikasi dengan menghitung probabilitas sebuah dokumen termasuk ke dalam kelas malicious dan non-malicious. Pada Gambar 3.5, $p(mal|X)$ adalah probabilitas doku-

men dengan fitur-fitur X tergolong *malicious*, dan $p(nonmal|X)$ adalah probabilitas dokumen dengan fitur-fitur X tergolong *nonmalicious*. Kedua nilai probabilitas tersebut dihitung sesuai dengan persamaan 2.8. Setelah kedua nilai probabilitas tersebut dihitung, dokumen akan diklasifikasi ke dalam kelas yang memiliki nilai probabilitas paling besar.

Untuk objek DecisionTreeClassifier klasifikasi dilakukan dengan cara mengikuti alur *node-node* dari *tree* yang telah terbentuk. Dokumen akan diklasifikasi sesuai dengan kelas dengan jumlah sampel terbanyak dari *node leaf* yang dicapai.



Gambar 3.6: Diagram alir proses klasifikasi oleh objek *DecisionTreeClassifier*.

Hasil klasifikasi dari kedua *classifier* tersebut adalah sebuah *list* sepanjang N, N adalah jumlah dokumen. Masing-masing elemen

pada N bernilai 0 (non-*malicious*) atau 1 (*malicious*). Dari hasil klasifikasi, dokumen-dokumen yang teridentifikasi *malicious* akan dipindahkan menuju direktori penampungan file-file *malicious* yang telah ditentukan pengguna melalui file konfigurasi. File-file dalam direktori tersebut akan dikelompokkan berdasarkan waktu klasifikasi.

Nama-nama dari file-file yang dipindahkan akan dirubah sesuai dengan format ”[index].php”. Contoh: misal direktori tempat penampungan file-file *malicious* bernama ”mal_dir/”. Maka dokumen pertama (index = 1) yang terklasikasi *malicious* pada tanggal 2016-08-17 jam 12:00:00 akan dipindahkan dari lokasi awal menuju mal_dir/2016-08-17_12:00:00/1.php. Nama dan lokasi (*path*) asli dari masing-masing file yang dipindahkan akan disimpan agar file-file tersebut bisa dikembalikan ke lokasi-lokasi asalnya apabila ternyata terbukti tidak *malicious*.

BAB IV

IMPLEMENTASI

Bab ini memberikan bahasan mengenai implementasi dari perancangan sistem yang dijelaskan pada bab sebelumnya.

4.1 Lingkungan Pembangunan Perangkat Lunak

Pembangunan perangkat lunak dilakukan pada lingkunan pengembangan sebagai berikut:

4.1.1 Lingkungan Perangkat Lunak

Adapun lingkungan perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

- Sistem operasi berupa Linux Ubuntu 16.04 64-bit.
- Python versi 2.7.12.
- PHP versi 7.0.8.

4.1.2 Lingkungan Perangkat Keras

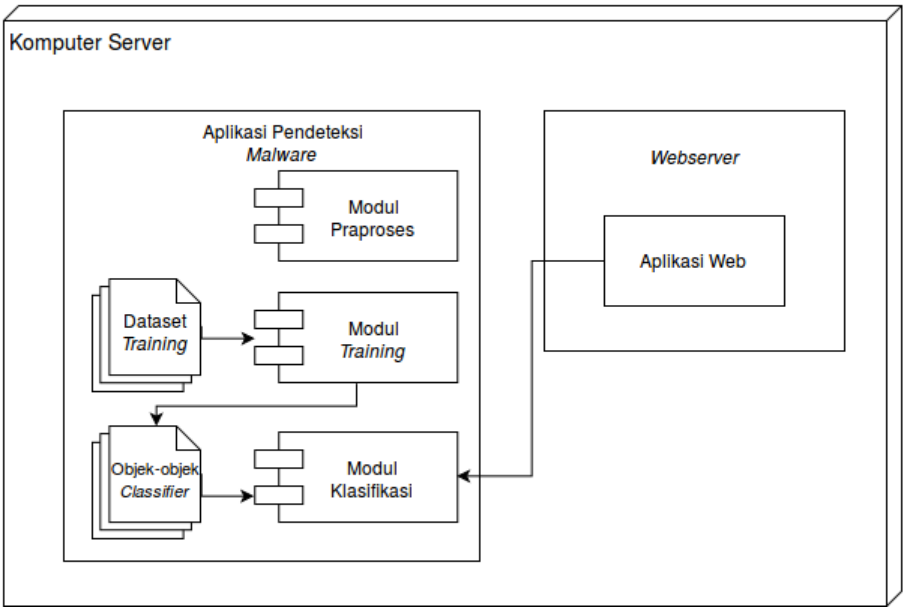
Spesifikasi perangkat keras yang digunakan pada lingkungan implementasi Tugas Akhir adalah sebagai berikut:

- *Processor* Intel® Core™ i5-2410M CPU @ 2.30GHz × 4.
- 4 GB DDR3 RAM.
- Media penyimpanan sebesar 750 GB.

4.2 Arsitektur Implementasi Aplikasi

Aplikasi akan dipasang dalam komputer server yang berisi aplikasi web target. Arsitektur dari implementasi ini dapat dilihat pada Gambar 4.1.

Diagram pada Gambar 4.1 menunjukkan komputer server yang berisi aplikasi pendeteksi *malware* dan sebuah perangkat lunak *webserver*. Aplikasi web target terletak pada direktori *webserver*, sedangkan aplikasi pendeteksi *malware* diletakkan di luar direktori tersebut. Terdapat beberapa komponen dalam aplikasi pendeteksi *malware*: modul praproses, modul *training*, dan modul klasifikasi. Selain itu disertakan juga dataset *training* awal dan objek-objek



Gambar 4.1: Diagram arsitektur implementasi aplikasi.

classifier hasil *training* menggunakan dataset tersebut. Dalam proses pendeteksian file *malicious*, file-file dari aplikasi web target akan dibaca oleh modul klasifikasi dari aplikasi pendeteksi *malware*. Modul klasifikasi akan menggunakan objek-objek *classifier* untuk mengklasifikasi file-file tersebut.

4.3 Implementasi Modul Praproses

Modul praproses diimplementasi dengan menggunakan dua bahasa pemrograman, yakni PHP dan Python. Bahasa pemrograman PHP digunakan untuk mengimplementasi proses pengambilan dan tokenisasi kode sumber PHP dari sebuah file. Proses tokenisasi tersebut dapat dilihat pada Pseudocode 4.1 dan 4.2.

```
Function is_long_string(S) begin
 if length(S) > SHORT_STRING_LENGTH then
```

```
  return true;
 else
  return false;
 end
end

Function is_base64_string(S) begin
 if base64_encode(base64_decode(S)) = S then
  return true;
 else
  return false;
 end
end

Function count_hex_literal(S) begin
 matches := regex_match("/\\\xleftarrow[a-fA-F0-9]{2}",S);
 return length(matches);
end
```

Pseudocode 4.1: Fungsi-fungsi pembantu proses tokenisasi

```
Function tokenize(file) begin
 isi := read(file);
 token_types ← "["T_EVAL, ""T_ECHO, ""T_VARIABLE, ""T_CLASS
     , "T_FUNCTION", ""T_CONST, "T_CONSTANT_ENCAPSED_STRING
     "];
 tokens := token_get_all(isi);
 i := 0;
 extracted_code := "";
 while (index < length(tokens)) do
  while (index < length(tokens)) do
   if (tokens[index].type = "T_OPEN_TAG") then
    break;
   end
   index := index + 1;
  end
  index := index+1
  while (index < length(tokens)) do
   if (tokens[index].type = "T_CLOSE_TAG") then
    break;
   end
```

```
   if (tokens[index].type in token_types) then
    if tokens[index].type = "T_CONSTANT_ENCAPSED_STRING"
        then
     if count_hex_literal(tokens[index].token) > 0 then
      extracted_code := extracted_code + "
          STRING_WITH_HEX_LITERAL" + line break;
     end
     else if is_long_string(tokens[index].token) then
      if is_base_64_string(tokens[index].token) then
       extracted_code := extracted_code + "
            BASE64_STRING_TOKEN" + line break;
      end
     else
      extracted_code := extracted_code + tokens[index].token
           + line break;
     end
    else
     extracted_code := extracted_code + tokens[index].token
         + line break;
    end
   end
   index := index + 1
  end
 end
 return extracted_code;
end
```

Pseudocode 4.2: Proses tokenisasi

Bagian modul praproses yang diimplementasi dengan bahasa PHP terdiri dari beberapa fungsi pembantu dan satu fungsi utama. Fungsi-fungsi pembantu tersebut adalah *is_long_string*, *is_base64_string*, dan *count_hex_literal*.

Fungsi *is_long_string* menerima sebuah *string* lalu menentukan apakah *string* tersebut termasuk *string* panjang atau bukan. Sebuah *string* dikatakan panjang apabila panjangnya melewati nilai *threshold* SHORT_STRING_LENGTH. Dalam implementasi ini, nilai SHORT_STRING_LENGTH yang digunakan adalah 100.

Fungsi kedua, *is_base64_string*, mengecek apakah sebuah *string*

ditulis dalam format *base64* yang valid. Pengecekan dilakukan dengan menerjemahkan *string* tersebut ke dalam bentuk biner dan mengkodekan ulang ke dalam bentuk *base64*. Apabila kode yang dihasilkan sama dengan *string* awal, maka *string* tersebut dikatakan sebagai *string base64* yang valid.

Kemudian fungsi pembantu yang ketiga, *count_hex_literal* bertujuan untuk menghitung jumlah karakter yang direpresentasikan dalam bentuk heksadesimal. Hal ini diimplementasikan dengan menggunakan *regular expression*.

Fungsi utama dari modul ini menerima sebuah file untuk ditokenisasi. Isi file tersebut akan dibaca dan disimpan ke dalam sebuah variabel. *Token-token* dari isi file akan diambil dengan menggunakan fungsi PHP *token_get_all*. Fungsi ini mengembalikan sebuah *array* yang berisi objek-objek *token*. Setiap blok kode PHP dimulai dengan *token* `<?php` atau `<?`. Kedua *token* tersebut tergolong jenis token T_OPEN_TAG. Oleh karena itu dilakukan perulangan pada *array* token untuk mencari *token* jenis T_OPEN_TAG. Apabila ditemukan, maka pembacaan *token* dilakukan hingga menemukan *token* ?>. Setiap *token* yang dibaca akan dicetak ke *stdout*. Dalam implementasi ini, *token-token* yang dicetak adalah yang termasuk dalam jenis-jenis token berikut:

- T_CLASS: nama kelas dan objek,
- T_VARIABLE: nama variabel,
- T_FUNCTION: nama fungsi,
- T_CONST: nama konstanta,
- T_CONSTANT_ENCAPSED_STRING: *string literal*

Token-token yang termasuk dalam T_CONSTANT_ENCAPSED_STRING akan melalui pengecekan dahulu dengan menggunakan fungsi-fungsi pembantu sebelum dicetak. Jika *token* tersebut adalah *string* panjang dan dalam format *base64* yang valid, maka yang dicetak adalah LONG_BASE64_STRING. Jika *token* tersebut termasuk *string* panjang, maka yang dicetak adalah LONG_NORMAL_STRING. Jika *token* tersebut memiliki satu atau lebih karakter da-

lam representasi heksadesimal, maka yang dicetak adalah STRING_-WITH_HEX_LITERAL. Ketika *token* ?> telah ditemukan maka pembacaan *token* untuk blok saat itu dihentikan. Pencarian blok kode sumber PHP berlanjut hingga mencapai akhir dari *array* token.

Bahasa pemrograman Python digunakan untuk membuat sebuah modul yang mengimplementasikan fungsi untuk menjalankan program PHP tersebut dan mengembalikan luarannya. Modul tersebut akan dapat digunakan oleh modul *training* dan klasifikasi.

4.4 Implementasi Modul *Training*

Modul *Training* diimplementasi dengan menggunakan bahasa Python. Implementasi dari modul ini menggunakan pustaka *scikit-learn* untuk melakukan vektorisasi dokumen dan membuat (*training*) model-model *classifier*. Pseudocode untuk implementasi modul *training* dapat dilihat pada Pseudocode 4.3.

```
Function read_dataset(configs):
 dataset := []; //variabel untuk menyimpan dokumen-dokumen
 target := [];  //variabel untuk menyimpan kelas dari
     dokumen (0=malicious, 1=non-malicious)
 remove_dataset_dir := false;

 if (configs['dataset'] is an archive) then
  make_directory('.extracted_dataset');
  extract(configs['dataset'], '.extracted_dataset');
  configs['dataset'] := '.extracted_dataset';
  remove_dataset_dir := true;
 end

 //baca file-file dari direktori dataset
 malicious_dir := configs['dataset'] + 'malicious';
 nonmalicious_dir := configs['dataset'] + 'nonmalicious';
 for each file in malicious_dir do
  dataset.append(tokenize(file))
  target.append(1)
 end
 for each file in nonmalicious_dir do
  dataset.append(tokenize(file))
  target.append(1)
```

```
 end
 if (remove_dataset_dir = true) then
  remove_directory('.extracted_dataset')
 end

 //buat objek CountVectorizer untuk mengubah dokumen menjadi
      matriks
 count_vectorizer := new CountVectorizer object;
 count_vectorizer.fit(dataset);
 count_matrix := count_vectorizer.transform(dataset);

 //buat objek TfidfTransformer untuk mengubah nilai-nilai
     pada count_matrix menjadi nilai tf-idf
 tfidf_transformer := new TfidfTransformer object;
 tfidf_transformer.fit(count_matrix);
 tfidf_matrix := tfidf_transformer.transform(count_matrix);

 return (tfidf_matrix, target, count_vectorizer,
     tfidf_transformer);

Function train_MultinomialNB(tfidf_matrix, target):
 classifier := new MultinomialNB object;
 classifier.fit(tfidf_matrix, target);
 return classifier;

Function train_DecisionTree(tfidf_matrix, target):
 classifier := new DecisionTreeClassifier object;
 classifier.fit(tfidf_matrix, target);
 return classifier

Function train(configs):
 tfidf_matrix, target, vectorizer, tfidf_transformer =
     read_dataset(configs['dataset']);
 mnb := train_MultinomialNB(tfidf_matrix, target);
 dt := train_DecisionTree(tfidf_matrix, target);
 export_object(vectorizer, 'classifier/vectorizer');
 export_object(tfidf_transformer, 'classifier/
     tfidf_transformer');
 export_object(mnb, 'classifier/MultinomialNB');
 export_object(dt, 'classifier/DecisionTree');
```

Pseudocode 4.3: Implementasi modul *training*

Modul ini terdiri dari beberapa fungsi pembantu dan satu fungsi utama. Fungsi *read_dataset* digunakan untuk mengambil dataset dari direktori yang ditentukan melalui objek konfigurasi *configs*. Fungsi ini menggunakan fungsi *tokenize* dari modul praproses untuk membaca dan tokenisasi kode sumber PHP dari sebuah file. Kode sumber dari masing-masing file akan disimpan sebagai entri dari sebuah objek *list*. Kemudian setelah semua file selesai dibaca, *list* kode sumber tersebut akan diubah menjadi matriks dengan menggunakan objek *CountVectorizer*. Matriks tersebut kemudian akan diubah lagi menjadi matriks tf-idf dengan menggunakan objek *TfidfTransformer*. Setelah semua itu selesai, fungsi ini akan mengembalikan 4 objek: matriks tf-idf, *list* berisi kelas dari masing-masing file, objek *CountVectorizer*, dan objek *TfidfTransformer*. Objek *CountVectorizer* dan *TfidfTransformer* juga dikembalikan karena objek-objek tersebut diperlukan juga dalam tahap klasifikasi.

Selain fungsi *read_dataset* terdapat dua fungsi pembantu lain yaitu *train_MultinomialNB* dan *train_DecisionTree*. Masing-masing fungsi tersebut menerima matriks tf-idf dan *list* kelas-kelas sebagai parameter masukan. Fungsi *train_MultinomialNB* akan membuat sebuah objek *MultinomialNB*. Sedangkan fungsi *train_DecisionTree* akan membuat sebuah objek *DecisionTreeClassifier*. Objek-objek tersebut akan dilatih dengan menggunakan data masukan lalu dikembalikan oleh fungsi.

Fungsi utama dari modul ini akan menjalankan fungsi-fungsi pembantu tersebut untuk membaca dataset, melakukan *training* objek-objek *MultinomialMB* dan *DecisionTreeClassifier*, kemudian mengekspor objek-objek tersebut ke dalam file-file. Fungsi ini menerima sebuah objek konfigurasi yang berisi parameter-parameter seperti letak direktori dataset.

4.5 Implementasi Modul Klasifikasi

Modul klasifikasi diimplementasi dengan menggunakan bahasa Python. Implementasi modul ini menggunakan fungsi *tokenize* dari modul praproses untuk melakukan tokenisasi kode sumber. Proses klasifikasi dilakukan dengan menggunakan objek-objek dari file-file luaran modul *training*. Implementasi dari modul klasifikasi dapat dilihat pada Pseudocode 4.4.

```
Function read_files(root_dir, documents, document_paths):
 for each file in root_dir do
  if (file is a directory) then
   read_files(file, documents, document_paths)
  else
   documents.append(tokenize(file)
   document_paths.append(path of file)
  end
 end

Function classify(configs):
 //baca semua objek-objek yang diperlukan untuk klasifikasi
 classifier := import_object(configs['classifier']);
 vectorizer := import_object(configs['vectorizer']);
 tfidf_transformer := import_object(configs['
     tfidf_transformer']);
 documents := []
 document_paths := []
 //baca dan tokenisasi file-file masukan secara rekursif
 read_files(configs['root_dir'], documents, document_paths)
 //ubah dokumen-dokumen masukan ke dalam bentuk matriks
 count_matrix := vectorizer.transform(documents);
 tfidf_matrix := tfidf_transformer.transform(count_matrix);
 //klasifikasi
 predictions := classifier.predict(tfidf_matrix);
 //pindahkan file-file yang terdeteksi sebagai malware ke
     direktori yang telah ditentukan
 malicious_dir := configs['malicious_dir'];
 //time_string akan menyimpan waktu sekarang dalam bentuk '
     YYYY-MM-DD_HH:MM:SS'
 time_string := current time;
```

```
index := 0;
original_paths_file = open_file(malicious_dir +
    current_time + 'original_paths.txt')
for each prediction in predictions:
 if (prediction = 1) then //jika prediksinya adalah
     malicious
  docname = time_string + '/' + index + '.php'
  move(document_paths[index], malicious_dir + docname)
  original_paths_file.write(document_paths[index] + ':␣' +
      docname + line break)
 end
 index := index + 1
end
```

Pseudocode 4.4: Implementasi modul klasifikasi

Modul klasifikasi terdiri dari satu fungsi utama dan satu fungsi pembantu. Fungsi *read_dataset* akan membaca file-file dari sebuah direktori secara rekursif. Fungsi ini menerima tiga parameter yakni lokasi direktori, sebuah objek list kosong bernama *documents*, dan objek *list* kosong lain bernama *document_paths*. Isi dari file yang dibaca akan dimasukkan sebagai entri dari *documents*, sedangkan lokasi file tersebut akan disimpan dalam *document_paths*. Setelah fungsi selesai dipanggil, kedua objek list tersebut masing-masing akan berisi isi dan lokasi dari file-file dalam direktori dan subdirektori yang dibaca.

Fungsi utama dari modul ini bertujuan untuk mengklasifikasi file-file dari sebuah direktori dengan menggunakan objek-objek hasil luaran dari modul *training*. Fungsi ini menerima sebuah objek konfigurasi yang berisi parameter-parameter yang diperlukan untuk menjalankan proses klasifikasi. Mula-mula file-file akan dibaca dengan menggunakan fungsi *read_dataset*. File-file tersebut akan dibaca dari direktori yang ditentukan pada objek konfigurasi. Kemudian objek-objek hasil modul *training* akan dibaca. Objek-objek tersebut adalah: *CountVectorizer, TfidfTransformer, dan MultinomialNB atau DecisionTreeClassifier*. *CountVectorizer* dan *TfidfTransformer* diguanakn untuk mengubah *list* dokumen menjadi ma-

triks tf-idf. Kemudian dari matriks tf-idf akan dilakukan klasifikasi menggunakan objek *classifier*. Dari dua objek *classifier* (*MultinomialNB* dan *DecisionTreeClassifier*), hanya satu objek yang akan digunakan untuk mengklasifikasi. Jenis *classifier* yang digunakan untuk mengklasifikasi ditentukan dalam objek konfigurasi.

Hasil klasifikasi berupa sebuah objek *list* yang berisi prediksi-prediksi kelas dari masing-masing file. File-file yang terindikasi *malicious* akan dipindahkan ke sebuah direktori yang dibuat khusus untuk menyimpan file-file *malicious*. Nama file yang dipindahkan akan diubah sesuai dengan format yang telah ditentukan untuk mencegah file-file dengan nama sama menumpuk satu sama lain. Lokasi beserta nama asli dari masing-masing file yang dipindahkan akan disimpan juga dalam sebuah file. Hal ini berguna apabila sebuah file dari direktori tersebut ternyata tidak *malicious* dan ingin dipindahkan ke lokasi awalnya.

(Halaman ini sengaja dikosongkan)

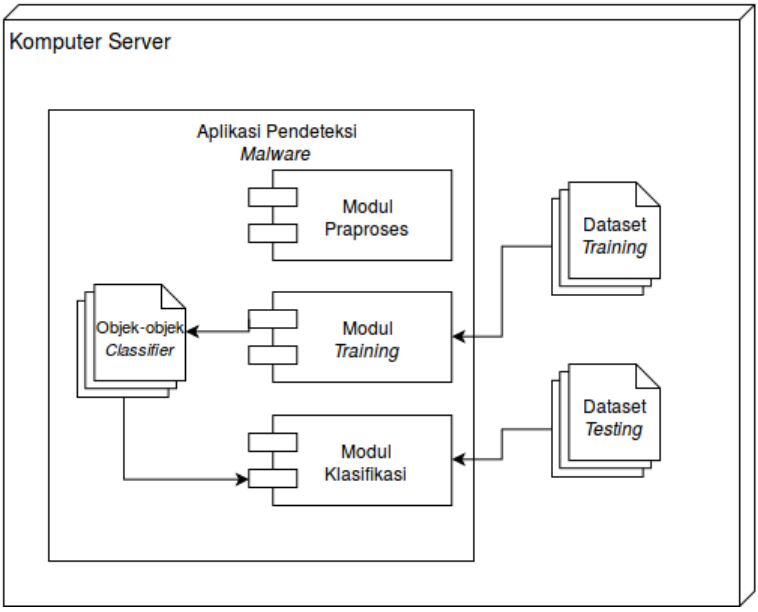
BAB V

UJI COBA DAN EVALUASI

Pada bab ini akan dibahas mengenai uji coba dan evaluasi dari aplikasi yang telah dibuat.

5.1 Lingkungan Uji Coba

Uji coba aplikasi dilakukan pada perangkat yang sama dengan perangkat yang digunakan pada tahap implementasi. Berbeda dengan pada saat implementasi, data uji coba tidak diambil dari aplikasi web target yang terpasang pada direktori *webserver*, namun dari kumpulan kode sumber-kode sumber yang telah dikumpulkan sebelumnya. Diagram pada gambar 5.1 menggambarkan lingkungan uji coba.



Gambar 5.1: Diagram arsitektur lingkungan uji coba.

5.2 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.

5.3 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 5.1.

Tabel 5.1: Skenario-skenario uji coba.

No	Jenis Uji Coba	Model *Classifier*	Hasil
1	Uji coba tahap *training*	Multinomial Naive Bayes	Spesifikasi model dan nilai *precision* & *recall*
2	Uji coba tahap *training*	Decision Tree	Spesifikasi model dan nilai *precision* & *recall*
3	Uji coba tahap *testing*	Multinomial Naive Bayes	Nilai *precision* & *recall*
4	Uji coba tahap *testing*	Decision Tree	Nilai *precision* & *recall*

Hasil uji coba tahap *training* akan menunjukkan detil model *classifier* yang dihasilkan, serta nilai *precision* dan *recall* dari ha-

sil klasifikasi model tersebut terhadap dataset *training*. Sedangkan uji coba tahap testing akan menunjukkan nilai *precision* dan *recall* dari hasil klasifikasi model *classifier* yang telah dibuat dalam tahap training terhadap dataset *testing*. Nilai *precision* didenifisikan dengan persamaan berikut:

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

dengan:

- P : nilai *precision*,
- TP: *true positive*, yakni jumlah kode *malicious* yang terklasifikasi sebagai *malicious*,
- FP: *false positive*, yakni jumlah kode non-*malicious* yang terklasifikasi sebagai *malicious*

Sedangkan *recall* didefinisikan sebagai berikut:

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

dengan:

- R : nilai *recall*,
- TP: *true positive*, yakni jumlah kode *malicious* yang terklasifikasi sebagai *malicious*,
- FN: *false negative*, yakni jumlah kode *malicious* yang terklasifikasi sebagai non-*malicious*

Berikut adalah detail masing-masing skenario:

Skenario I: Uji coba tahap *training* dengan *Multinomial Naive Bayes*

Sebanyak 427 file *malicious* dan 500 file non-*malicious* dari dataset *training* akan digunakan sebagai data masukan dalam proses training menggunakan modul *training*. Modul training akan dikonfigurasi sehingga menghasilkan model *classifier Multinomial Naive Bayes* dari data masukan. Setelah itu mo-

del *classifier* yang dihasilkan akan diuji coba dengan dataset yang sama menggunakan modul klasifikasi.

Skenario II: Uji coba tahap *training* dengan *Decision Tree*

Seperti pada skenario I, 427 file *malicious* dan 500 file non-*malicious* dari dataset *training* akan digunakan sebagai data masukan dalam proses training. Modul training akan dikonfigurasi sehingga menghasilkan model *classifier Decision Tree* dari data masukan. Hasil *model classifier* tersebut akan diuji coba dengan dataset yang sama.

Skenario III: Uji coba tahap *testing* dengan *Multinomial Naive Bayes*

Uji coba dilakukan setelah model *classifier Multinomial Naive Bayes* selesai dibuat. Sebanyak 218 file *malicious* dan 500 file non-*malicious* dari dataset *testing* akan digunakan sebagai data masukan dalam proses *testing* menggunakan modul klasifikasi.

Skenario IV: Uji coba tahap *testing* dengan *Decision Tree*

Uji coba ini dilakukan setelah model *classifier Decision Tree* selesai dibuat. Sebanyak 218 file *malicious* dan 500 file non-*malicious* dari dataset *testing* akan digunakan sebagai data masukan dalam proses *testing* menggunakan modul klasifikasi.

5.4 Hasil Uji Coba

Berikut adalah hasil uji coba dari masing-masing skenario:

5.4.1 Hasil Skenario I: Uji coba tahap *training* dengan *Multinomial Naive Bayes*

Proses *training* dari Skenario I menghasilkan sebuah objek *classifier Multinomial Naive Bayes* yang memiliki atribut feature_log_prob seperti ditunjukkan oleh Tabel 5.2 dan Tabel 5.3. Seperti yang telah dijelaskan pada bab sebelumnya, nilai-nilai pada *feature_log_prob_* adalah bobot dari masing-masing *token* yang menunjukkan seberapa besar pengaruh kemunculan *token* tersebut terhadap pro-

babilitas suatu dokumen untuk masuk ke suatu kelas tertentu. Ada dua jenis nilai *feature_log_prob_*, yakni nilai *feature_log_prob* suatu *token* untuk kelas *malicious* dan nilai *feature_log_prob* suatu *token* untuk kelas *non-malicious*. Nilai-nilai inilah yang nantinya digunakan untuk menghitung probabilitas masing-masing kelas (*malicious* dan non-*malicious*) untuk masing-masing file masukan pada saat uji coba.

Tabel 5.2: Sepuluh token dengan nilai feature_log_prob tertinggi untuk kelas *malicious*.

No	Token	Nilai dalam feature_log_prob_ (log_{10})
1	echo	-5.79046
2	_post	-5.90729
3	globals	-6.40187
4	file	-6.7021
5	long_normal_string	-6.93612
6	eval	-7.01603
7	_get	-7.08006
8	base64_decode	-7.11216
9	_request	-7.12115
10	dir	-7.13348

Uji coba dari *model classifier Multinomial Naive Bayes* dengan menggunakan dataset *training* menghasilkan nilai-nilai seperti yang ditunjukkan oleh Tabel 5.4

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).

Tabel 5.3: Sepuluh token dengan nilai feature_log_prob tertinggi untuk kelas non-*malicious*.

No	Token	Nilai dalam feature_log_prob_ ($\log_{10}$)
1	this	-5.68568
2	echo	-6.62656
3	args	-6.8012
4	jetpack	-6.92215
5	blog_id	-6.98191
6	__	-7.0025
7	add_action	-7.03369
8	atts	-7.13127
9	id	-7.15479
10	false	-7.13348

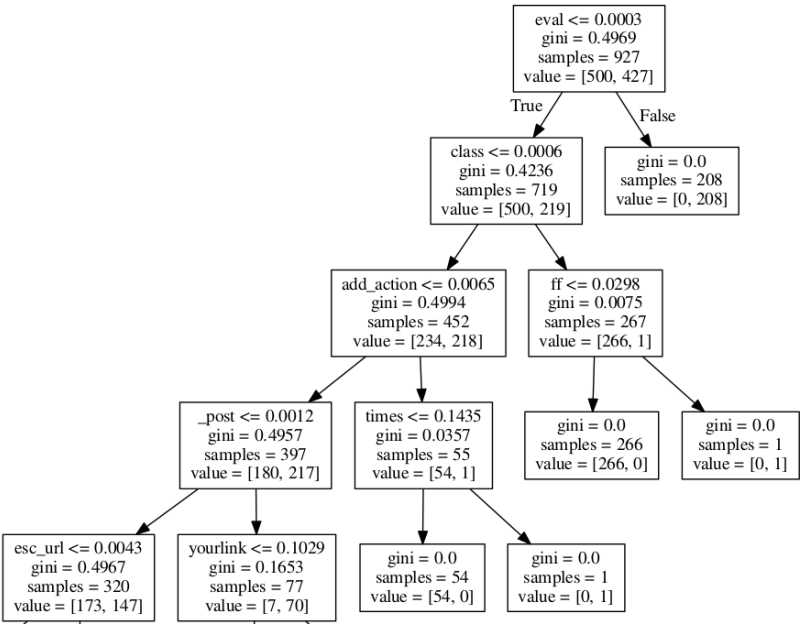
Tabel 5.4: Hasil prediksi dataset *training* dengan *classifier Multinomial Naive Bayes*.

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

5.4.2 Hasil Skenario II: Uji coba tahap *training* dengan *Decision Tree*

Proses *training* dari Skenario II menghasilkan sebuah objek *classifier Decision Tree* seperti yang diilustrasikan Gambar 5.2. Ilustrasi menunjukkan *tree* hingga kedalaman 5 tingkat saja karena keterbatasan tempat. Hasil model *tree* sebenarnya memiliki kedalaman 20 tingkat.

Masing-masing *node* pada *tree* yang ditunjukkan oleh Gambar 5.2 memiliki beberapa informasi. Informasi pertama adalah jenis *token* yang dicek beserta nilai tf-idf pembatasnya. Contohnya pada



Gambar 5.2: Ilustrasi model *classifier Decision Tree* hasil proses *training* hingga kedalaman 5.

node teratas, nilai tf-idf dari token `eval` akan dicek. Apabila nilai tersebut kurang dari atau sama dengan 0.0003 maka pengecekan berlanjut ke *node* cabang sebelah kiri, namun apabila tidak, pengecekan berlanjut ke *node* sebelah kanan.

Informasi selanjutnya adalah nilai *gini impurity* dari dataset pada *node* tersebut. Nilai *gini impurity* dihitung sesuai dengan persamaan 2.9. Semakin besar nilai *gini impurity*, maka perbandingan jumlah sampel *malicious* dan non-*malicious* pada dataset tersebut semakin berimbang. Nilai *gini impurity* minimal adalah 0, yakni ketika semua sampel pada dataset termasuk dalam satu kelas saja (tidak ada sampel dari kelas lain).

Informasi ketiga dan keempat adalah jumlah total sampel dari

dataset dan jumlah sampel untuk masing-masing kelas. Pada *node* teratas, "`samples= 927`" menunjukkan bahwa dataset pada node tersebut memiliki 927 sampel. Dari 927 sampel tersebut, 500 sampel tergolong non-*malicious* dan 427 sisanya *malicious*.

Uji coba dari *model classifier Decision Tree* dengan menggunakan dataset *training* menghasilkan nilai-nilai seperti yang ditunjukkan oleh Tabel 5.5

Tabel 5.5: Hasil prediksi dataset *training* dengan *classifier Decision Tree*.

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

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).

5.4.3 Hasil Skenario III: Uji coba tahap *testing* dengan *Multinomial Naive Bayes*

Uji coba dari *model classifier Multinomial Naive Bayes* dengan menggunakan dataset *testing* menghasilkan nilai-nilai seperti yang ditunjukkan oleh Tabel 5.6

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).

Tabel 5.6: Hasil prediksi dataset *testing* dengan *classifier Multinomial Naive Bayes*.

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

5.4.4 Hasil Skenario IV: Uji coba tahap *testing* dengan *Decision Tree*

Uji coba dari *model classifier Decision Tree* dengan menggunakan dataset *testing* menghasilkan nilai-nilai seperti yang ditunjukkan oleh Tabel 5.7

Tabel 5.7: Hasil prediksi dataset *testing* dengan *classifier Decision Tree*.

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

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).

5.5 Evaluasi Hasil

Dari hasil tahap *training* dapat dilihat token-token mana saja yang dianggap penting bagi masing-masing model *classifier* untuk

menentukan apakah suatu file mengandung kode sumber *malicious* atau tidak. Bagi model *classifier Multinomial Naive Bayes*, apabila suatu file memiliki banyak token `_post`, `globals`, dan seterusnya seperti yang ditunjukkan dalam Tabel 5.2, maka kemungkinan besar file tersebut adalah file *malicious*. Namun apabila token-token tersebut muncul dalam jumlah yang sedikit, atau token-token yang ditunjukkan dalam Tabel 5.3 muncul dalam jumlah yang lebih banyak, maka kemungkinan besar file tersebut adalah file non-*malicious*. Perlu diperhatikan bahwa ada *token-token* seperti `echo` yang memiliki nilai *feature_log_prob_* tinggi tidak hanya untuk satu kelas, melainkan keduanya. Hal ini menyebabkan pengaruh *token-token* tersebut dalam menentukan apakah sebuah file *malicious* atau tidak menjadi tidak terlalu besar.

Untuk model *Decision Tree*, dapat dilihat bahwa token `eval` dianggap sebagai *token* yang terpenting dan merupakan token yang paling pertama diperiksa. Apabila nilai tf-idf dari token ”eval” lebih besar daripada 0.0003 maka *classifier* akan langsung mengklasifikasikan file tersebut sebagai file *malicious*. Apabila tidak, maka *classifier* akan memeriksa token-token lainnya dengan urutan sesuai yang ditunjukkan oleh Gambar 5.2.

Hasil klasifikasi dari kedua *classifier* tersebut dirangkum dalam Tabel 5.8. Dapat dilihat bahwa ternyata hasil klasifikasi menggu-

Tabel 5.8: Nilai *precision* dan *recall* dari masing-masing *classifier*.

	Multinomial Naive Bayes – Training	Decision Tree – Training	Multinomial Naive Bayes – Testing	Decision Tree – Testing
Precision	98,10%	92,22%	83,41%	72,35%
Recall	84,77%	100,00%	83,03%	97,24%

nakan *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*.

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

Pada bab ini akan diberikan kesimpulan yang diambil dari pengerjaan Tugas Akhir ini dan hasil uji cobanya, serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Berdasarkan proses perancangan, implementasi, dan pengujian terhadap aplikasi yang dibuat dalam Tugas Akhir ini, dapat diambil kesimpulan bahwa:

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, dan *string literal*), membuat *model classifier* dan menggunakan model *classifier* tersebut untuk mengklasifikasi file-file lain.
2. Model *Multinomial Naive Bayes* yang dihasilkan dari tahap *training* mengklasifikasi dengan cara menghitung probabilitas masing-masing kelas berdasarkan nilai tf-idf masing-masing *token*. Tiga *token* yang paling berpengaruh untuk mengklasifikasi dokumen menurut model yang dihasilkan adalah `echo`, `_post`, dan `globals`.
3. Model *Decision Tree* yang dihasilkan dari tahap *training* membuat sebuah *tree* / pohon keputusan yang berisi kondisi-kondisi di masing-masing *node*-nya. *Token* yang dianggap paling berpengaruh untuk membedakan file *malicious* dan non-*malicious* menurut model yang dibangun adalah *token* `eval`.
4. 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 *re-*

call hingga 97%. Model *classifier Multinomial Naive Bayes* cenderung lebih presisi daripada *Decision Tree*, namun *Decision Tree* dapat lebih banyak mendeteksi file-file *malicious* daripada *Multinomial Naive Bayes*.

6.2 Saran

Saran yang dapat diberikan dari Tugas Akhir ini adalah:

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* [7]. 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 *Multinomial Naive Bayes*. Algoritma-algoritma tersebut dapat diterapkan dan dibandingkan performanya dengan algoritma-algoritma yang telah diterapkan pada Tugas Akhir 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. URL `http://doi.acm.org/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. URL `http://doi.acm.org/10.1145/361219.361220`.

[4] scikit-learn. `http://scikit-learn.org`, . Diakses: 2015-12-26.

[5] Harry Zhang. The optimality of naive bayes. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*. AAAI Press, 2004.

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

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

[8] Wordpress. `https://www.wordpress.org`, . Diakses: 2016-12-23.

[9] Usage of content management system for websites. https://w3techs.com/technologies/overview/content_management/all/, . Diakses: 2016-12-23.

[10] Wordpress plugin. https://wordpress.org/plugins/about/, . Diakses: 2016-12-23.

[11] The security state of wordpress' top 50 plugins. https://www.checkmarx.com/wp-content/uploads/2013/06/The-Security-State-of-WordPress-Top-50-Plugins.pdf, . Diakses: 2016-12-24.

[12] Compromised web servers and web shells - threat awareness guidance. https://www.us-cert.gov/ncas/alerts/TA15-314A. Diakses: 2016-12-23.

A.1 Kode sumber *tokenizer*

```
<?php
define("SHORT_STRING_LENGTH", 100);
define("BASE64_STRING_TOKEN",
↪ "LONG_BASE64_STRING");
define("LONG_STRING_TOKEN",
↪ "LONG_NORMAL_STRING");
define("STRING_WITH_HEX_LITERAL",
↪ "STRING_WITH_HEX_LITERAL");

//argv[1] = path dataset
//argv[2] = path file spesifikasi token
if (isset($argv[2])) {
    $tokens_file = fopen($argv[2], 'r');
}
else {
    $tokens_file = fopen('tokens.txt', 'r');
}

function is_long_string($token) {
    if (strlen($token[1]) > SHORT_STRING_LENGTH)
↪ {
        return true;
    }
    return false;
}
function is_base64_string($token) {
    $original = substr($token[1],1,-1);
    $recoded =
↪ base64_encode(base64_decode($original));
    if ($recoded === $original) {
        return true;
```

```
27      }
28      return false;
29  }
30  function count_hex_literals($token) {
31      return preg_match_all("/\\\x[a-fA-F0-9]{2}/",
    ↪  $token[1]);
32  }
33
34  $choosen_tokens = array();
35  $token = fgets($tokens_file);
36  while($token) {
37      $token = trim($token);
38      $choosen_tokens[$token] = true;
39      $token = fgets($tokens_file);
40  }
41
42  $content = file_get_contents($argv[1]);
43  $tokens = token_get_all($content);
44  $num_tokens = count($tokens);
45  $index = 0;
46  $extracted_code = '';
47  while($index < $num_tokens) {
48      while($index < $num_tokens) {
49          if (is_array($tokens[$index]) &&
    ↪  token_name($tokens[$index][0]) ==
    ↪  "T_OPEN_TAG") {
50              break;
51          }
52          $index++;
53      }
54      $index++;
55      while($index < $num_tokens) {
```

```
56          if (is_array($tokens[$index]) &&
    ↪  token_name($tokens[$index][0]) ==
    ↪  "T_CLOSE_TAG") {
57              break;
58          }
59          if (is_array($tokens[$index]) && ar-
    ↪  ray_key_exists(token_name($tokens[$index][0]),
    ↪  $choosen_tokens)) {
60              if (token_name($tokens[$index][0]) ==
    ↪  "T_CONSTANT_ENCAPSED_STRING") {
61                  if
    ↪  (count_hex_literals($tokens[$index]) > 0) {
62                      $extracted_code .=
    ↪  STRING_WITH_HEX_LITERAL . PHP_EOL;
63                  }
64                  else if
    ↪  (is_long_string($tokens[$index])) {
65                      if
    ↪  (is_base64_string($tokens[$index])) {
66                          $extracted_code .=
    ↪  BASE64_STRING_TOKEN . PHP_EOL;
67                      }
68                      else {
69                          $extracted_code .=
    ↪  LONG_STRING_TOKEN . PHP_EOL;
70                      }
71                  }
72                  else {
73                      $extracted_code .=
    ↪  $tokens[$index][1] . PHP_EOL;
74                  }
75              }
76              else {
```

```
77                      $extracted_code .=
    ↪  $tokens[$index][1] . PHP_EOL;
78                  }
79              }
80              $index++;
81          }
82      }
83      echo $extracted_code;
84
85      ?>
```

A.2 Kode sumber modul praproses

```
from subprocess import check_output
import re

config_file = open('tokenizer_config', 'r')
config_regex = re.compile(r'(\w+) *= *"(.+)"')
comment_regex = re.compile(r'^\s*#')
configs = {}
#path ke script tokenizer
configs['tokenizer'] = 'tokenizer2.php'
#path ke file daftar token
configs['tokens'] = 'tokens.txt'
for line in config_file:
    is_comment = comment_regex.search(line)
    if (is_comment):
        continue
    match = config_regex.search(line)
    if (match):
        configs[match.groups()[0].lower()] =
↪   match.groups()[1]


def tokenize(filename):
    try:
        file = open(filename, 'r')
        file.close()
    except IOError:
        raise
    return check_output(['php',
↪   configs['tokenizer'], filename,
↪   configs['tokens']])
```

A.3 Kode sumber modul *training*

```
#!/usr/bin/python
if (__name__ == "__main__"):
    print ("Menginisialisasi...")

from sklearn.feature_extraction.text import
 ↪ CountVectorizer
from sklearn.feature_extraction.text import
 ↪ TfidfTransformer
from tokenizer import tokenize
import os
import os.path
import shutil
import tarfile
from subprocess import check_output
import re
import pickle

class Dataset:
    def __init__(self, data, target):
        self.data = data
        self.target = target

def parse_configs(config_path):
    config_file = open(config_path, 'r')
    config_regex = re.compile(r'(\w+) *=
 ↪ *"(.+)"')
    comment_regex = re.compile(r'^\s*#')
    #dictionary yang menyimpan key-value config
    configs = {}
    #path ke folder dataset
    configs['dataset'] = 'dataset'
```

```
29      #path ke folder classifier
30      configs['classifier_folder'] = 'classifier'
31      for line in config_file:
32          is_comment = comment_regex.search(line)
33          if (is_comment):
34              continue
35          match = config_regex.search(line)
36          if (match):
37              configs[match.groups()[0].lower()] =
    ↪  match.groups()[1]
38
39      return configs
40
41
42  def export_object(configs, obj, name):
43      file = open(configs['classifier_folder'] +
    ↪  name, 'wb')
44      pickler = pickle.Pickler(file, 2)
45      pickler.dump(obj)
46      file.close()
47
48
49  def read_dataset(configs):
50      dataset = []
51      target = []
52
53      if (os.path.isdir(configs['dataset'])):
54          dataset_folder = configs['dataset']
55          if (not dataset_folder.endswith('/')):
56              dataset_folder = dataset_folder + '/'
57
58          mal_folder = dataset_folder +
    ↪  'malicious/'
```

```
59          nonmal_folder = dataset_folder +
    ↪ 'nonmalicious/'
60
61          #baca source code malware
62          for filename in os.listdir(mal_folder):
63              dataset.append(tokenize(mal_folder +
    ↪ filename))
64              target.append(1)
65
66          #baca source code non-malware
67          for filename in
    ↪ os.listdir(nonmal_folder):
68              dataset.append(tokenize(nonmal_folder
    ↪ + filename))
69              target.append(0)
70
71      elif
    ↪ (configs['dataset'].endswith('.tar.gz')):
72          dataset_tar =
    ↪ tarfile.open(configs['dataset'], 'r:gz')
73          if (not
    ↪ os.path.isdir('.extracted_dataset')):
74              os.mkdir('.extracted_dataset')
75
    ↪ dataset_tar.extractall('.extracted_dataset')
76          dataset_folder = '.extracted_dataset/'
77
78          mal_folder = dataset_folder +
    ↪ 'malicious/'
79          nonmal_folder = dataset_folder +
    ↪ 'nonmalicious/'
80
81          #baca source code malware
```

```
82          for filename in os.listdir(mal_folder):
83              dataset.append(tokenize(mal_folder +
    ↪  filename))
84              target.append(1)
85
86          #baca source code non-malware
87          for filename in
    ↪  os.listdir(nonmal_folder):
88              dataset.append(tokenize(nonmal_folder
    ↪  + filename))
89              target.append(0)
90
91          shutil.rmtree('.extracted_dataset')
92
93      #buat matrix term (n-gram) count
94      count_vectorizer =
    ↪  CountVectorizer(encoding='utf-8',
    ↪  decode_error='replace',
    ↪  ngram_range=(configs['ngram'],configs['ngram']))
95      count_matrix =
    ↪  count_vectorizer.fit_transform(dataset)
96
97      #buat matriks tfidf dan tfidf transformer
98      tfidf_transformer =
    ↪  TfidfTransformer().fit(count_matrix)
99      tfidf_matrix =
    ↪  tfidf_transformer.transform(count_matrix)
100
101
102
103     return (Dataset(tfidf_matrix, target),
    ↪  count_vectorizer, tfidf_transformer)
104
```

```
105
106 def train_MultinomialNB(dataset):
107     from sklearn.naive_bayes import MultinomialNB
108     classifier = MultinomialNB().fit(dataset.data,
    ↪   dataset.target)
109     return classifier
110
111 def train_DecisionTree(dataset):
112     from sklearn import tree
113     classifier =
    ↪   tree.DecisionTreeClassifier().fit(dataset.data,
    ↪   dataset.target)
114     return classifier
115
116 def to_number(value):
117     try:
118         return float(value)
119     except ValueError:
120         return False
121
122 if __name__ == "__main__":
123     configs = parse_configs('training_config')
124     print ("Membaca dataset...")
125     dataset, vectorizer, tfidf_transformer =
    ↪   read_dataset(configs)
126     print ("Training: Multinomial Naive
    ↪   Bayes...")
127     mnb = train_MultinomialNB(dataset)
128     print ("Training: Decision Tree...")
129     dt = train_DecisionTree(dataset)
130
131     if (not
    ↪   configs['classifier_folder'].endswith('/')):
```

```
132         configs['classifier_folder'] =
    ↪ configs['classifier_folder'] + '/'
133
134     export_object(configs, vectorizer,
    ↪ 'vectorizer')
135     export_object(configs, tfidf_transformer,
    ↪ 'tfidf_transformer')
136
137     export_object(configs, mnb, 'MultinomialNB')
138     export_object(configs, dt, 'DecisionTree')
139
140
141
```

A.4 Kode sumber modul klasifikasi

```
#!/usr/bin/python

import pickle
import datetime
from os.path import isdir
from os.path import abspath
from os import listdir
from os import mkdir
from shutil import move
import re
import argparse
from tokenizer import tokenize

def append_slash(dir_path):
    if (dir_path[-1] != '/'):
        dir_path += '/'
    return dir_path

def import_object(path):
    try:
        file = open(path, 'rb')
        unpickler = pickle.Unpickler(file)
        return unpickler.load()
    except IOError:
        return False

def parse_configs(config_path):
    configs = {}
    configs['classifier'] = False
    configs['vectorizer'] =
    ↪ 'classifier/vectorizer'
```

```
31      configs['tfidf_transformer'] =
    ↪  'classifier/tfidf_transformer'
32      configs['root_path'] = False
33      configs['malicious_dir'] = False
34      config_file = open(config_path, 'r')
35      config_regex = re.compile(r'(\w+) *=
    ↪  *"(.+)"')
36      comment_regex = re.compile(r'^\s*#')
37
38      for line in config_file:
39          is_comment = comment_regex.search(line)
40          if (is_comment):
41              continue
42          match = config_regex.search(line)
43          if (match):
44              configs[match.groups()[0].lower()] =
    ↪  match.groups()[1]
45
46      return configs
47
48  def read_files(root_path, documents = [],
    ↪  document_paths = []):
49      files = listdir(root_path)
50      for file in files:
51          if (isdir(root_path + file)):
52              read_files_recursive(root_path + file
    ↪  + '/', documents, document_paths)
53          else:
54              documents.append(tokenize(root_path +
    ↪  file))
55              document_paths.append(root_path +
    ↪  file)
56
```

```
57      return (documents, document_paths)
58
59  if __name__ == '__main__':
60      #parse configs
61      configs = parse_configs('classify_config')
62      if (configs['root_path'] == False or
    ↪   not(isdir(configs['root_path'])) ):
63          print 'Error: root_path tidak valid'
64          exit(1)
65      if (configs['malicious_dir'] == False or
    ↪   not(isdir(configs['malicious_dir']))):
66          print 'Error: path malicious_dir tidak
    ↪   valid'
67          exit(1)
68      if (configs['classifier'] == False):
69          print 'Error: file classifier belum
    ↪   dispesifikasikan'
70          exit(1)
71
72
73      configs['root_path'] =
    ↪   append_slash(configs['root_path'])
74      configs['malicious_dir'] =
    ↪   append_slash(configs['malicious_dir'])
75
76      #baca objek-objek hasil training
77      classifier =
    ↪   import_object(configs['classifier'])
78      vectorizer =
    ↪   import_object(configs['vectorizer'])
79      tfidf_transformer =
    ↪   import_object(configs['tfidf_transformer'])
80
```

```
81      if (classifier == False):
82          print 'Error: file classifier "' +
    ↪   configs['classifier'] + '" tidak dapat
    ↪   dibaca'
83          exit(1)
84      if (vectorizer == False):
85          print 'Error: file vectorizer "' +
    ↪   configs['vectorizer'] + '" tidak dapat
    ↪   dibaca'
86          exit(1)
87      if (tfidf_transformer == False):
88          print 'Error: file tfidf_transformer "' +
    ↪   configs['tfidf_transformer'] + '" tidak dapat
    ↪   dibaca'
89          exit(1)
90
91      documents = []
92      document_paths = []
93
94      read_files(configs['root_path'], documents,
    ↪   document_paths)
95
96      #vectorisasi dokumen
97      count_matrix =
    ↪   vectorizer.transform(documents)
98      tfidf_matrix =
    ↪   tfidf_transformer.transform(count_matrix)
99
100     #klasifikasi
101     predictions =
    ↪   classifier.predict(tfidf_matrix)
102
```

```
103     #pindahkan file-file yang terdeteksi malicous
    ↪   ke malicious_dir
104     malicious_dir = configs['malicious_dir']
105     original_paths = {}
106     current_time =
    ↪   datetime.datetime.now().isoformat('_')
107     mkdir(malicious_dir + current_time)
108     original_paths_txt = open(malicious_dir +
    ↪   current_time + 'original_paths.txt', 'w')
109     index = 0
110     for prediction in predictions:
111         print(document_paths[index])
112         print(prediction)
113         if (prediction == 1):
114             docname = current_time + '/' +
    ↪   str(index) + '.php'
115             move(document_paths[index],
    ↪   malicious_dir + docname)
116             origi-
    ↪   nal_paths_txt.write(document_paths[index] + ':
    ↪   ' +docname + '\n')
117         index += 1
118     original_paths_txt.close()
```

BIODATA PENULIS

Fransiskus Gusti Ngurah Dwika Setiawan biasa dipanggil Dwika, lahir di Denpasar pada tanggal 14 Juli tahun 1994. Penulis merupakan anak kedua dari dua bersaudara. Penulis menempuh pendidikan SDN 2 Ubung Denpasar (2000-2006), SMPN 10 Denpasar (2006-2009), SMAN 4 Denpasar (2009-2012). Minat penulis terhadap komputer dimulai sejak saat penulis duduk di bangku SD, hingga akhirnya mengikuti klub komputer pada saat duduk di bangku SMA. Selama SMA penulis pernah mengikuti lomba-lomba pemrograman yang diselenggarakan di berbagai universitas. Tertarik dengan komputer dan pemrograman, penulis memutuskan untuk melanjutkan studi di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nompember Surabaya. Selama menjadi mahasiswa, penulis pernah mengikuti kompetisi pemrograman Indonesia National Programming Contest yang diselenggarakan oleh Binus University dan masuk dalam top 50 dalam kompetisi tersebut. Penulis juga pernah mengikuti kompetisi-kompetisi keamanan jaringan dalam bentuk *capture the flag* yang diselenggarakan di internet. Penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK) untuk pengerjaan Tugas Akhirnya. Alamat e-mail yang dapat digunakan untuk menghubungi penulis adalah fgn.dwikasetiawan@gmail.com.