IDENTIFIKASI MALWARE ANDROID MENGGUNAKAN PENDEKATAN ANALISIS HIBRID DENGAN DEEP LEARNING

Raden Budiarto Hadiprakoso[1)], Nurul Qomariasih[2)], dan Ray Novita Yasa[3)]
[1, 2, 3)]Rekayasa Kriptografi, Politeknik Siber dan Sandi Negara
Jalan H. USA, Ciseeng, Bogor
e-mail: raden.budiarto@poltekssn.ac.id[1)], nurul.qomariasih@poltekssn.ac.id[2)], ray.novita@poltekssn.ac.id[3)]

ABSTRAK

Android merupakan sistem operasi mobile yang paling populer digunakan saat ini. Bagaimana pun dibalik kepopuleran ini muncul ancaman penyebaran malware pada platform Android. Pada pertengan tahun 2021 peneliti keamanan dari Quick Heal Security Labs mendeteksi setidaknya ada delapan aplikasi di Google Play Store yang disusupi oleh malware Joker. Malware ini dapat secara sembunyi-sembunyi membuat ponsel korbannya berlangganan dan membayar konten premium tanpa sepengetahuan korban. Untuk itu, deteksi malware Android ini sangat penting untuk menjaga keamanan dan privasi pengguna. Bagaimana pun karena proses identifikasi malware yang semakin rumit, maka perlu digunakan pendekatan deep learning untuk klasifikasi malware. Makalah ini menggabungkan fitur analisis statis dan dinamis dari aplikasi malware dan aplikasi bukan malware. Fitur dinamis diambil dari panggilan API pada aplikasi sedangkan fitur statis didapatkan melalui permission, system call dan intent. Model deep learning dengan arsitektur LSTM (Long Short-Term Memory) dikembangkan untuk mengidentifikasi malware. Hasil pengujian pada data uji menunjukkan model yang dikembangkan memiliki akurasi 98,7%, recall 97,9% dan presisi 99,6% serta skor F1 98,7%..

***Kata Kunci**: malware android, deteksi malware, machine learning, LSTM*

ABSTRACT

Android is the most popular mobile operating system used today. However, behind this popularity comes the threat of spreading malware on the Android platform. In mid-2021 security researchers from Quick Heal Security Labs detected at least eight apps in the Google Play Store that were compromised by the Joker malware. This malware can secretly make the victim's phone subscribe and pay for premium content without the victim's knowledge. For this reason, Android malware detection is very important to maintain user security and privacy. However, due to the increasingly complex malware identification process, it is necessary to use a deep learning approach for malware classification. This paper combines the features of static and dynamic analysis of malware applications and non-malware applications. Dynamic features are retrieved from API calls on the application while static features are obtained through permissions, system calls and intents. A deep learning model with LSTM (Long Short-Term Memory) architecture was developed to identify malware. The test results on the test data show that the developed model has 98.7% accuracy, 97.9% recall and 99.6% precision and 98.7% for F1 score.

***Keywords**: android malware, malware detection, deep learning, LSTM*

I. PENDAHULUAN

Android merupakan sistem operasi *mobile* yang paling banyak digunakan pada saat ini. Menurut Sundar Pichai, CEO Google menyatakan terdapat sekitar 2 miliar perangkat Android diaktifkan pada bulannya [1]. Pada pertengahan 2021 berdasarkan laporan dari *Global Stat* Counter, sistem operasi Android meraih 72,74% dari total pasar sistem operasi seluler [2]. Hal ini yang menjadikannya sistem operasi seluler yang paling banyak digunakan bahkan melampaui sistem operasi Windows. Bagaimana pun, popularitas yang dimiliki Android tidak selalu berdampak positif bagi penggunanya. Salah satu dampak negatif adalah masalah *malware* yang terpadat pada sistem operasi Android. Sistem Android adalah target utama *malware* seluler karena sistem operasi Android memungkinkan pengguna untuk menginstal aplikasi yang diunduh dari pasar pihak ketiga. Bahkan dengan berbagai mekanisme proteksi keamanan seperti *play protection*, masih terdapat berbagai laporan keberadaan *malware* pada *Google Play Store*. Pada bulan Juni 2021 peneliti keamanan dari *Labs Quick Heal Security* mendeteksi setidaknya ada delapan aplikasi di *Play Store* yang disusupi oleh *malware* Joker [3]. *Malware* ini dapat secara sembunyi-sembunyi membuat ponsel korbannya berlangganan membayar layanan premium tanpa sepengetahuan korban.

Berdasarkan berbagai kejadian tersebut, masih perlu penelitian lebih lanjut tentang untuk identifikasi *malware* Android. Pendekatan yang berbeda telah disarankan dalam berbagai penelitian sebelumnya dengan maksud untuk menemukan *malware*. Pendekatan ini dapat dikategorikan ke dalam analisis statis dan analisis dinamis [4]. Dalam pendekatan berbasis statis, prosedur analisis diselesaikan tanpa mengeksekusi sampel *malware* terlebih dahulu. Pada deteksi *malware* berbasis statis biasanya mengidentifikasi *malware* dengan menyesuaikan pola program yang

dianalisis berdasarkan basis data *signature* (atribut) dari aplikasi *malware* yang telah diketahui sebelumnya [5]. Strategi ini bergantung pada asumsi bahwa kasus *malware* dapat didefinisikan melalui sebuah pola dan ciri tertentu. Kelemahan prosedur analisis statis ini adalah karena pola *signature byte* berasal dari *malware* yang sudah dikenal maka untuk menghindarinya juga mudah. Penyerang dapat menggunakan pendekatan *code obfuscation* [6] untuk mengecoh program pendeteksi *malware*.

Di sisi lain, analisis berbasis dinamis adalah cara untuk menemukan *malware* dengan melakukan sampel *malware* ini di lingkungan yang dikendalikan seperti dalam emulator atau mesin virtual. Kinerja pendekatan ini bergantung pada kapasitas perangkat keras untuk mendeteksi perilaku berbahaya sebagai *runtime*. Selain itu perlu menyediakan lingkungan yang ideal untuk mengeksekusi kode berbahaya. Kelemahan dari investigasi berbasis dinamis memakan banyak waktu dan sumber daya (prosesor & memori) karena menggunakan lingkungan dengan kondisi khusus untuk pengujian sampel *malware* [7]. Kendala ini mengilhami peneliti untuk mengajukan strategi analisis berbasis hibrid untuk hasil yang lebih efektif. Melalui pendekatan analisis hibrid ini diharapkan dapat diperoleh manfaat dari kedua pendekatan tersebut.

Berangkat dari latar belakang tersebut. makalah ini mengusulkan dan menyelidiki pendekatan baru yang memanfaatkan keunggulan *deep learning* yang digabungkan pendekatan analisis hibrid. Hal ini dilakukan dengan tujuan mencapai deteksi akurasi *malware* yang lebih baik. Selanjutnya, kami akan melakukan eksperimen untuk mengevaluasi pendekatan berbasis *deep learning* dengan membandingkan dengan berbagai algoritma *machine learning* dalam hal identifikasi *malware* pada Android. Kontribusi dari makalah ini dapat dirangkum sebagai berikut: (1) Kami menggunakan dataset dari kombinasi data dari hasil analisis statis dan analisis dinamis [8]. (2) Kami menggunakan model *deep learning* untuk analisis data yang bersifat runtutan yaitu dengan arsitektur *Recurrent Neural Network* (RNN). Kami menggunakan *Long Short-Term Memory* (LSTM) dikarenakan arsitektur LSTM cocok untuk digunakan pada data dinamis dan dapat mengatasi permasalahan *vanishing gradient* [9].

Adapun sisa dari penjabaran pada makalah ini adalah sebagai berikut. Bagian 2 menyajikan penelitian-penelitian terkait. Kemudian penjelasan terkait metodologi di bagian 3 dan dilanjutkan pada bagian 4 yang menunjukkan hasil eksperimen beserta pembahasannya. Pada bagian akhir yakni bagian 5, kami menutup dengan kesimpulan dan saran untuk penelitian berikutnya.

II. PENELITIAN TERKAIT

Pada beberapa tahun terakhir, *Machine Learning* (ML) mulai digunakan untuk mengembangkan sistem cerdas dengan cara melatih mesin untuk membuat keputusan. Dengan menggunakan dataset sebagai input, ML dapat mengidentifikasi data baru yang memiliki kemiripan pola dengan dataset inputan. Terdapat berbagai algoritma klasifikasi ML dapat digunakan untuk membangun model ML, dan masing-masing algoritma memiliki kelebihan tergantung pada dataset yang digunakan. Osusanwo dkk., dalam penelitian mereka membandingkan pengklasifikasi ML yang termasuk dalam kelompok *Supervised* [9]. Algoritma ML yang digunakan dalam penelitian ini meliputi *Decision Tree*, *Naïve Bayes* (NB), *Support Vector Machine* (SVM), *Random Forest* (RF) dan *Artifisial Neural Networks* (ANN). Proses analisis data menggunakan alat bantu *Waikato Environment for Knowledge Analysis* (WEKA). Hasilnya adalah SVM merupakan algoritma yang memiliki akurasi dan presisi terbaik.

Dalam penelitian [10], peneliti melakukan percobaan beberapa algoritma ML: NB *Multi-layer Perceptron* (MLP), J48, *K-Nearest Neighbors* (KNN), dan RF untuk mengidentifikasi *malware* pada Android. Hasilnya pengklasifikasi BN dan RF memiliki hasil terbaik dalam hal deteksi *malware*. Meskipun true-positive rate (TPR) yang diperoleh NB dan RF mencapai 94,97%, namun dalam hal pendeteksian *malware* Android terbaru, KNN lebih unggul dengan memperoleh TPR tertinggi, 83,67%.

Pada tahun 2018, Chen dkk. [11] melakukan penelitian dalam identifikasi *malware*. Chen membangun kerangka kerja deteksi *malware* yang disebut StromDroid. Kerangka kerja ini yang mencakup enam algoritma pengklasifikasi ML, SVM, *Decision Tree* (C4.5), ANN, NB, dan K-NN. Untuk menghitung kinerja algoritma ML, Chen membaginya menjadi dua kategori, kategori pertama, yang berisi fitur izin dan panggilan API sensitif, dan kategori kedua berisi fitur izin, panggilan API sensitif, urutan, dan perilaku dinamis. Pada kategori pertama, performa terbaik diperoleh K-NN dan ANN dengan rata-rata akurasi di atas 92,00%. Sedangkan pada kategori kedua, SVM dan K-NN memperoleh akurasi tertinggi dengan nilai rata-rata 92,85% dan 94,18%. Semua algoritma berjalan di bawah satu menit selama waktu eksekusi, kecuali ANN, yang membutuhkan waktu lebih lama. Secara umum, dalam penelitian ini menyatakan K-NN merupakan algoritma klasifikasi dengan performa terbaik berdasarkan sumber daya komputasi yang paling kecil.

Makalah [12] menegaskan hubungan penting antara *malware* dan atribut pada *file* manifes Android. Sementara itu, Fauzia Idrees menyatakan bahwa izin (*permission*) dan *file* manifes Android berguna secara akurat untuk mengklasifikasikan *malware* [13]. Menggunakan analisis statis Proses analisis statis umumnya menggunakan

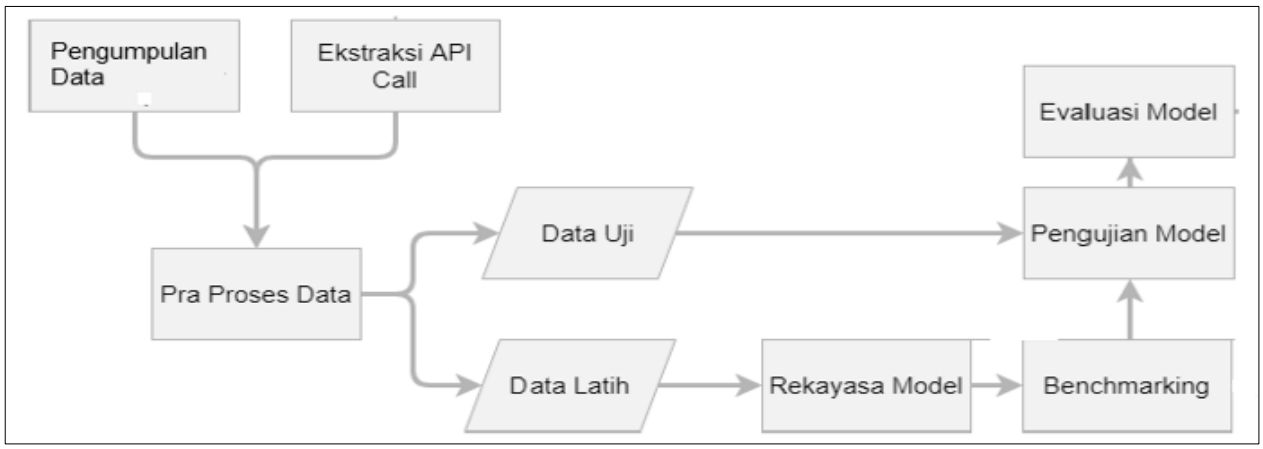
ekstraksi fitur, seperti filter *intent* [14], *system command*, dan *fingerprinting* [15]. Sebagai perbandingan, analisis dinamis atau perilaku dapat dilakukan dengan melacak *system call* atau urutan API *call* [16] atau menganalisis instruksi kode program [17].

Pada tahun 2017, penelitian dilakukan oleh Fereidooni. dkk. [18] melakukan pendeteksian *malware* berbasis ML menggunakan analisis statis pada aplikasi android. Sebuah aplikasi, *uniDroid*, dibuat untuk mengekstrak informasi yang terkandung dalam aplikasi android. Pada aplikasi ini, beberapa algoritma klasifikasi ML dilatih untuk melihat performa yang terbaik pada akurasi dan kecepatannya. Hasilnya menunjukkan bahwa penelitian mendapatkan nilai akurasi 97% untuk mendeteksi *malware* dengan algoritma ANN. Pada penelitian [19], dilakukan analisis dinamis, yang menggunakan *syscall* untuk menangkap dan menganalisis perilaku jejak panggilan sistem yang dibuat oleh setiap aplikasi saat menjalankan aplikasi. Tingkat akurasi 86% diperoleh dengan menggunakan algoritma *decision tree* dan akurasi 88% menggunakan algoritma RF.

Dalam penelitian yang dilakukan oleh Rahman dkk., deteksi *malware* dilakukan pada sistem operasi Android dengan pendekatan hibrid untuk meningkatkan akurasi dalam mendeteksi *malware* dan melengkapi kelemahan metode analisis statis dan dinamis [20]. Penelitian ini termasuk membahas bagaimana mengembangkan sistem pendeteksi *malware* yang dapat mendeteksi berbagai jenis *malware*. Mereka juga menjelaskan cara mendeteksi *malware* sebelum proses instalasi, memeriksa aplikasi seluler untuk kemungkinan ancaman *malware*, dan memperingatkan pengguna Android tentang *malware* setelah proses pengunduhan, dari beberapa tahap. Tahap pertama adalah mengekstrak *string* dari aplikasi android dan menjelajahi *file* manifes android. Langkah selanjutnya adalah memisahkan *string* kata kunci dari manifes android. Tahap ketiga adalah mengklasifikasikan *malware* dan aplikasi yang aman menggunakan kata kunci dan *string* sebagai input. Tahap terakhir adalah mengidentifikasi aplikasi berbahaya dan aplikasi yang aman. Hasil *dari* penelitian ini menunjukkan bahwa SVM merupakan algoritma yang paling cocok dibandingkan dengan algoritma lainnya, dan model yang diusulkan mendapatkan akurasi hingga 85,51%.

III. METODE

Skema penelitian ini dibagi menjadi beberapa langkah, seperti yang diilustrasikan pada Gambar 1. Secara garis besar meliputi tahap-tahap berikut: (1) pengumpulan data, (2) pengolahan data, (3) membuat model, (4) pengujian dan evaluasi model. Berikut penjelasan dari masing-masing tahapan.



Gambar. 1. Tahap-tahap penelitian

Tahap pertama adalah pengumpulan data. Dataset yang digunakan dalam penelitian ini dibagi menjadi dua jenis data yaitu data latih dan data uji. Data latih merupakan data untuk proses pemodelan klasifikasi. Data model menggunakan dataset sumber terbuka [8]. Data uji merupakan data yang digunakan untuk implementasi model klasifikasi yang sudah final. Data *mawar* tersebut perlu dilakukan proses ekstraksi API Call untuk dijadikan dataset sebagian masukan untuk model klasifikasi. Data yang digunakan belum sesuai dengan persyaratan untuk dilakukan pemrosesan agar dapat digunakan untuk masukan *deep learning* [21].Untuk mengumpulkan data, kami mengambil sampel aplikasi *malware* dari *VirusShare* dan APK bukan *malware* yang diambil melalui APK setelah melewati tahap pemindaian *malware* menggunakan Cuckoo Sandbox. Kami mengumpulkan *malware* dan aplikasi bukan *malware* dalam jumlah yang seimbang dengan 5450 aplikasi legal dan 5550 aplikasi *malware*. Jumlah ini

diperlukan untuk melatih model *deep learning* dan menghindari *overfitting* pada model.

Setelah 11000, aplikasi *malware* dan aplikasi jinak dikumpulkan. Kami mengekstrak fitur (atribut) statis dalam bentuk *permission* manifes, *command signature*, dan *intent*. Proses ekstraksi fitur ini dilakukan secara otomatis menggunakan *framework SaveDroid*. Sebanyak 100 fitur diekstraksi melalui hasil analisis statis ini. Setelah semuanya terkumpul, kami mengumpulkan data melalui analisis dinamis. Pada tahap ekstraksi API dilakukan proses ekstraksi *raw malware*. Proses ekstraksi menggunakan alat bantu *cuckoo sandbox*. *Malware* dimasukkan ke *cuckoo sanbox* dengan menggunakan API dari *cuckoo sandbox*. Hasil analisis dari *cuckoo sandbox* diambil dengan menggunakan API *cuckoo sandbox* dengan format data JSON. Data hasil analisis tersebut dipilah untuk mendapatkan urutan pemanggilan API saat analisis dinamis. Ketika telah mendapatkan daftar pemanggilan API, data tersebut kemudian diubah dalam bentuk indeks sesuai dengan label indeks pada penelitian. Fitur yang diekstrak adalah urutan panggilan API untuk 100 panggilan pertama. Total 200 atribut yang dikumpulkan ini dijabarkan seperti pada tabel 1.

TABEL I
ATRIBUT YANG DIEKSTRAK DARI SAMPEL APLIKASI

Atribut	Kategori	Jumlah
Pemanggilan API	Dinamis	100
Permission	Statis	50
System Command	Statis	25
Intent	Statis	25
Total		200

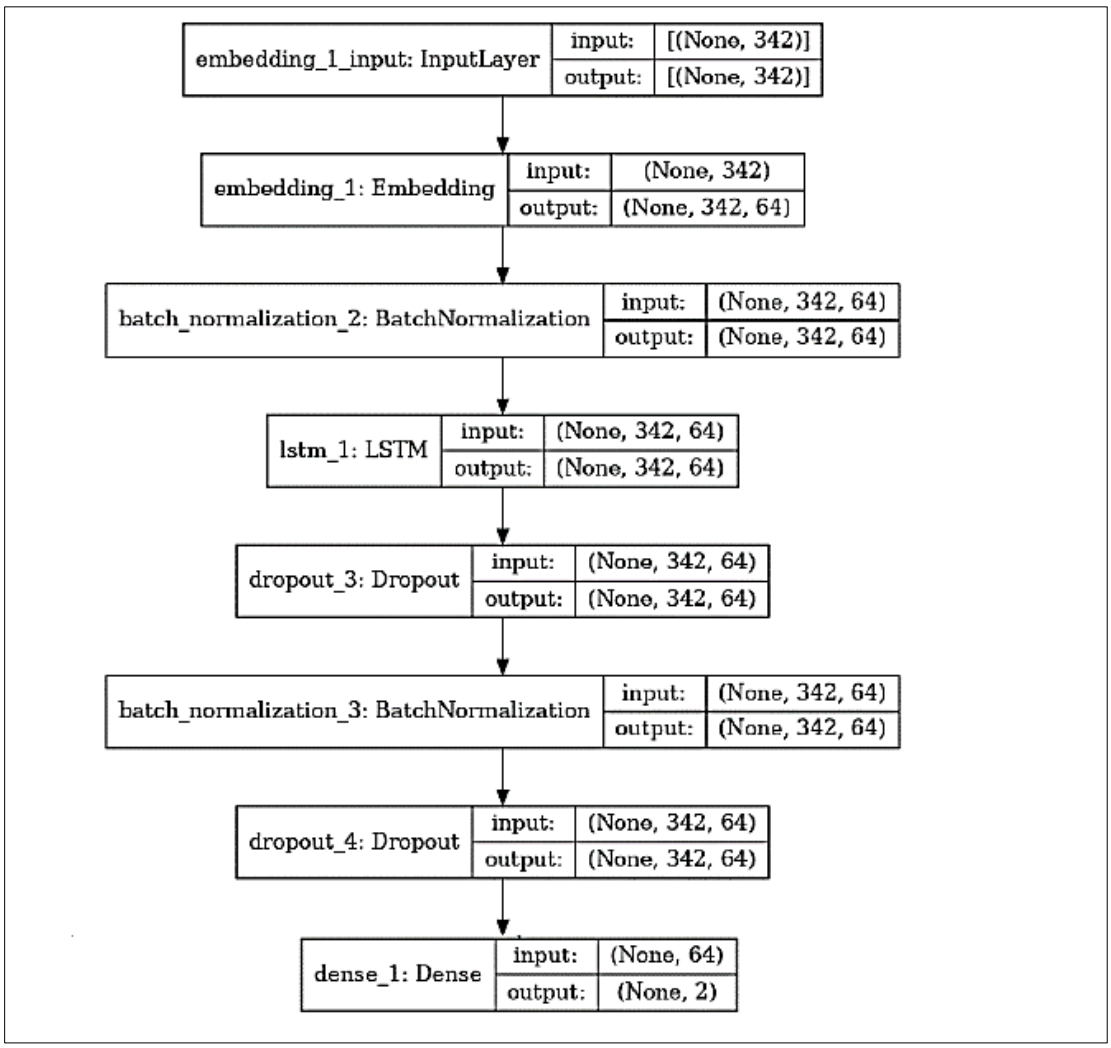
Pada tabel 1 diketahui terdapat 4 kategori dari atribut yang diekstrak. Pertama adalah pemanggilan API, kategori ini merupakan hasil dari analisis dinamis karena diperlukan eksekusi aplikasi untuk mengetahui aplikasi tersebut mengonsumsi API dari mana saja. Melalui pemanggilan API ini dapat diketahui aplikasi mengambil atau memberikan data dari atau ke pihak mana, oleh sebab itu hal ini dapat membedakan aplikasi yang berbahaya dan yang jinak. Melalui atribut *permission* yang terdapat pada *file* manifest.xml kita dapat mengetahui aplikasi diberikan izin untuk mengakses data apa saja. Hal ini termasuk izin untuk mengakses data kontak, histori telepon, koordinat GPS (*global positioning system*), kamera dan sebagainya. Fitur *command* menunjukkan perintah pada level sistem operasi yang digunakan pada aplikasi. Semua fitur ini dapat membedakan antara aplikasi berbahaya (*malware*) dan aplikasi jinak.

Setelah mendapatkan 200 atribut dari 11000 *malware* dan aplikasi jinak, kami melakukan langkah kedua pra-pemrosesan data. Pada tahap ini, kami menelusuri semua data yang telah dikumpulkan. Karena proses pengambilan data dilakukan secara otomatis dengan mengandalkan aplikasi *Cuckoo* maka terdapat beberapa nilai kosong yang kemungkinan disebabkan oleh kegagalan aplikasi. Pada tahap pra-pemrosesan ini kami menghapus 487 data yang kosong atau tidak valid. Kemudian seluruh atribut ini ditandai dengan nilai biner dan disimpan dalam file CSV (*comma-separated value*). Dari 200 atribut ini, kami menambahkan kolom terakhir yang merupakan kelas aplikasi, 1 untuk aplikasi *malware*, dan 0 untuk aplikasi jinak. Sebelum melanjutkan ke proses selanjutnya, kami membagi data yang telah dikumpulkan untuk pelatihan dan pengujian, 80% untuk pelatihan model, 20% untuk pengujian.

Pada proses rekayasa model kami menggunakan model jaringan syaraf tiruan dengan arsitektur berbasis LSTM (*Long Short-Term Memory*). Model ini digunakan dalam proses klasifikasi *malware* dengan memanfaatkan runtutan pemanggilan *API Call.* Arsitektur LSTM berkaitan dengan *Long Term Memory* (LTM) dan *Short Term Memory* (STM), dan untuk membuat perhitungan sederhana dan efektif. Hasil dari model ini berupa model yang dapat melakukan prediksi kelas dari *malware* yang diujikan. Model LSTM terdiri atas parameter *embeding, batch_normalization,* LSTM, *dropout, attention,* serta *dense*. Diagram dari model LSTM dapat dilihat pada Gambar 2.

Model yang dibangun terdiri atas 342 fitur *sequence* dari pemanggilan *API call*. Input tersebut masuk ke dalam layer *embedding* dengan *input dimension* 800, *output dimension* 64 dan panjang input 342. Kemudian dilakukan proses normalisasi dengan masuk ke layer *batch_normalization.* Setelah itu masuk ke dalam layer LSTM untuk dilakukan proses *training*. Setelah keluar dari *layer* LSTM kemudian masuk ke dalam layer *dropout.* Pada layer *dropout* dilakukan pembuangan dari neuron secara acak untuk meminimalisasi terjadinya *overfitting* pada model klasifikasi. Setelah itu dilakukan proses normalisasi kembali. Penempatan *batch normalization* dilakukan sebelum *dropout layer*. Hal ini dilakukan untuk menghindari ketidakharmonisan antara *dropout layer* dengan *bach normalization* yang dapa mempengaruhi hasil *learning* [22]. *Dropout* digunakan untuk menghindari model yang *overfit.* Hasil keluaran dari layer *dropout* berupa 64 unit neuron. Kemudian masuk ke dalam layer *dense* di mana 64 unit neuron dipangkas menjadi 2 unit neuron. Setiap unit

dari keluar *layer dense* merepresentasikan kelas *malware* dan non-*malware*.



Gambar. 2. Model *deep learning* yang digunakan

Tahap selanjutnya adalah evaluasi dan *benchmarking* model. Pada tahap ini, kami mengevaluasi model berdasarkan dataset pelatihan. Untuk tujuan perbandingan algoritme, kami membuat tiga set data pelatihan independen. Dataset pertama hanya berisi fitur 100 statis; dataset kedua hanya menggunakan 100 fitur dinamis. Dataset ketiga menggunakan fitur dinamis dan statis. Kami juga membandingkan model hasil pelatihan dengan beberapa algoritme pembelajaran mesin tradisional. Algoritma yang digunakan untuk perbandingan adalah *Support Vector Machine* (SVM), *Naive Bayes* (NB), *K-Nearest Neighbor* (K-NN) dan *Random Forest* (RF). Dari perbandingan tersebut, kami akan mendapatkan algoritma yang dianggap lebih efektif dalam mengklasifikasikan aplikasi apakah itu termasuk *malware* atau bukan.

Pengujian *cross* 5-fold *validation* digunakan dalam percobaan yang dilakukan. Melalui prosedur ini dataset akan dikelompok menjadi 5 bagian dengan besaran jumlah yang sama, kemudian akan dilakukan metrik pengukuran terhadap semua bagian tersebut satu per satu. Sebagai metrik pengukuran, kami menggunakan akurasi, presisi, *recall*, dan skor F1. Persamaan 1-4 menunjukkan rumus yang digunakan untuk menentukan akurasi, presisi, *recall*, dan skor F1.

$$Akurasi = TP + TN/TP + FP + FN + TN \quad (1)$$
$$Presisi = TP/TP + FP \quad (2)$$
$$Recall = TP/TP + FN \quad (3)$$
$$\text{F1 } Score = 2 * (Recall * Precision) \,/\, (Recall + Precision) \quad (4)$$

Keterangan: TP (*True Positive*), TN (*True Negative*), FP *(False Positive*), FN (*False Negative)*

IV. HASIL DAN PEMBAHASAN

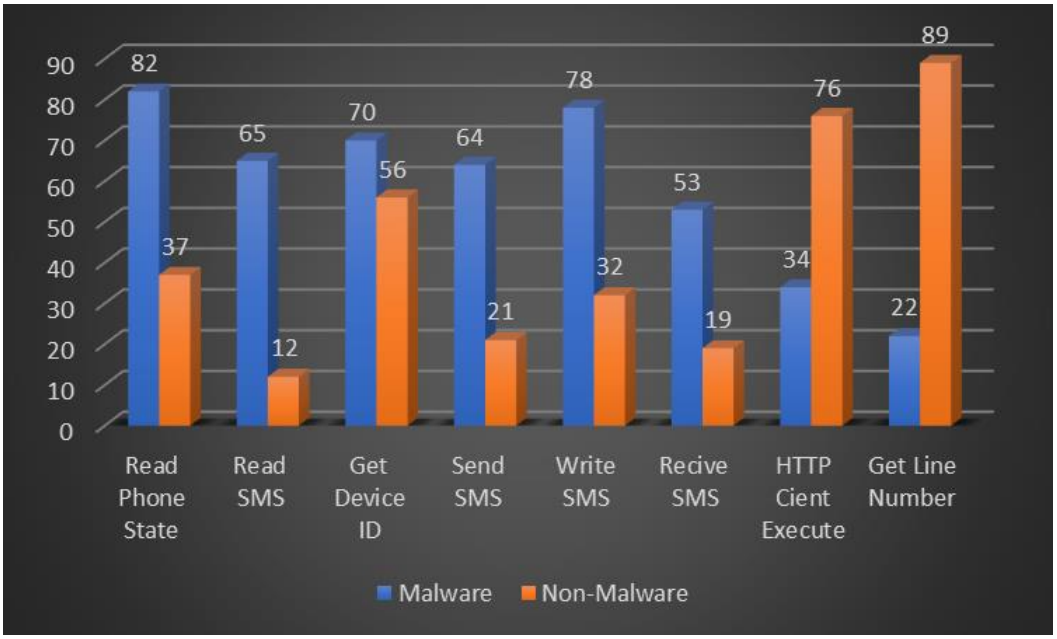
Setelah data terkumpul, kami menggunakan alat bantu *Google Colab* untuk memproses data. Pada tahap pra-pemrosesan data, kami mendapatkan 487 nilai kosong dan tidak valid pada kumpulan data yang dikumpulkan. Semua nilai dihapus pada saat ini. Nilai yang hilang ini terutama disebabkan oleh data yang rusak saat mengumpulkan data dalam lingkungan berbasis kerangka kerja. Hasil tersebut menyisakan 10.513 aplikasi untuk diproses ke tahap selanjutnya. Total ini terdiri dari 5.421 aplikasi jinak dan 5.092 *malware*. Sebelum memasuki tahap membagi data, kami membagi data 80% dan 20% untuk validasi dan pengujian.

Sebagai perbandingan, kami menyajikan hasil data *training* pada berbagai algoritma ML. Berdasarkan hasil tersebut, model *artifisial neural network* (ANN) yang kami rancang berhasil mencapai hasil deteksi terbaik pada akurasi 100%, diikuti oleh algoritma *Random Forest* (RF) dengan 98,56%. Dari tabel 2 diketahui bahwa algoritma ANN memiliki rata-rata akurasi paling tinggi dibandingkan dengan algoritma ML lainnya. Model DNN yang menggunakan fitur statis dan dinamis juga terbukti sedikit lebih baik daripada model jaringan syaraf tiruan yang hanya menggunakan fitur statis (ANN-S) dan dinamis (ANN-D). Namun, kelemahan model DNN adalah waktu pelatihan yang lebih lama. Kami memasang data ke model menggunakan *optimizer* Adam selama 100 *epoch*. Untuk menguji algoritma ML, *library* Scikit-learn dan bahasa pemrograman Python digunakan. Hasil model pengujian model menggunakan data *training* dapat dilihat pada tabel II.

TABEL II
HASIL PENGUJIAN MODEL PADA DATA LATIH

Algoritma	Akurasi	Recall	Presisi
ANN	1.0000	1.0000	1.0000
ANN-S	0.9642	0.9521	0.9682
ANN-D	0.8478	0.8923	0.8903
SVM	0.9394	0.9524	0.9684
NB	0.7386	0.7767	0.7197
RF	0.9556	0.9812	0.9878
K-NN	0.9206	0.9119	0.9124

Algoritma NB tidak menghasilkan skor akurasi yang tinggi karena cenderung bekerja lebih efektif pada data latih yang lebih sedikit. Algoritma ini juga memiliki waktu latensi terpendek karena dataset pelatihan hanya disimpan dalam memori dan digunakan kembali selama prediksi. Hal ini menghasilkan waktu pelatihan yang singkat tetapi waktu pengujian yang lebih lama. Cara kerja yang sama juga berlaku untuk algoritma K-NN. Oleh karena itu, algoritma ini memiliki waktu pelatihan yang relatif singkat. Algoritma K-NN menunjukkan hasil yang cukup baik, diduga karena distribusi datanya yang non-linier karena algoritma ini terkenal dengan klasifikasi data yang non-linear. Selama tahap pengujian kami juga mengamati fitur-fitur apa yang paling membedakan antara aplikasi *malware* dan aplikasi jinak. Hasil perbandingan fitur ini disajikan pada gambar 3.



Gambar 3. Perbandingan fitur antara aplikasi *malware* dan bukan *malware*

Berdasarkan gambar 3, dapat dilihat bahwa dua fitur yang paling membedakan *malware* dengan non-*malware* adalah fitur *read phone state* dan *get device id*, di mana *read phone state* adalah fitur yang paling umum digunakan pada *malware*, diikuti oleh fitur *get device id*. Selain itu, fitur *read* SMS (*permission*) dan *write* SMS juga menunjukkan perbedaan yang signifikan karena kedua fitur ini juga lebih sering ditemukan pada *malware*. Khususnya, fitur membaca SMS dapat digunakan untuk merekam pesan singkat, dan *get device id* dapat digunakan untuk menangkap nomor telepon pengguna, yang mungkin berarti bahwa aplikasi *malware* lebih cenderung mencuri informasi sensitif pengguna.

Selain itu, kita dapat melihat bahwa eksekusi kelas *Http Client* terjadi lebih sering pada aplikasi jinak. Tepatnya, fitur *Http Client* yang dapat dieksekusi digunakan secara teratur untuk menanyakan layanan HTTP secara *online*. Hasil ini menarik karena kami menganggap aplikasi berbahaya terhubung ke server HTTP untuk mentransfer lebih banyak data. Namun demikian, berdasarkan pengujian yang dilakukan, aplikasi *malware* cenderung menggunakan cara lain (pesan SMS dan komunikasi soket) untuk mengirimkan data secara diam-diam.

Pada tahap pengujian yang terakhir dilakukan evaluasi model dan perbaikan model. Di sini kami menguji model yang telah dilatih sebelumnya menggunakan dataset uji. Model *overfit* dapat terjadi karena terlalu rumit berdasarkan model *fitting* pada data *training*. Nilai akurasi model yang sempurna menunjukkan salah satu gejala dari hal ini. Untuk mencegah *overfit*, kami menyederhanakan model dengan mengurangi jumlah neuron dan lapisan tersembunyi. Selain itu, kami juga bereksperimen dengan menggunakan lapisan *dropout* untuk mencegah model dari *overfitting*. Pada akhirnya, kami membatalkan penggunaan *dropout* karena cenderung hasil yang tidak konsisten. Hasil dari model *tuning* (penyesuaian *hyper*-parameter dari model) ini disajikan pada tabel III.

TABLE III
HASIL PENYESUAIAN PARAMETER MODEL PADA DATA UJI

Jumlah Neuron, Layer	Akurasi	Recall	Presisi	Skor F1
8,2	0.958	0.980	0.974	0.977
8,4	0.968	0.953	0.960	0.956
16,4	0.987	0.979	0.996	0.987
16,8	0.978	0.960	0.992	0.976
32,4	0.975	0.952	0.964	0.958
32,8	0.972	0.960	0.959	0.959
64, 4	0.962	0.976	0.952	0.964
64, 8	0.966	0.955	0.998	0.976

Tabel III menunjukkan bahwa model *deep neural network* dengan empat layer dengan 16 neuron pada skenario pengujian ketiga. Menggunakan parameter ini model menghasilkan nilai akurasi 98,7%, *recall* 97,9% dan presisi 99,6% serta skor F1 98,7%. Hasil ini menunjukkan hasil yang cukup konsisten dari hasil *training* sebelumnya

dengan sedikit penurunan menunjukkan bahwa model telah fit. Berdasarkan hasil pengujian di tabel III menunjukkan bahwa jumlah ideal neuron dan layer adalah 16 neuron dan 4 layer. Hal ini ditunjukkan oleh nilai akurasi, *recall*, presisi dan skor F1 yang lebih tinggi dibandingkan dengan yang lainnya. Penggunaan nilai yang lebih tinggi kemungkinan akan menghasilkan model *overfit* dan memperlambat waktu pelatihan atau prediksi. Sebaliknya penggunaan jumlah neuron dan layer di bawah angka ini akan menyebabkan model yang *underfit* serta kurang optimal dalam akurasi maupun skor F1.

V. KESIMPULAN

Setelah melalui tahap pengujian pada penelitian ini maka dapat disimpulkan bahwa pengujian dengan model *deep learning* dan arsitektur LSTM menunjukkan hasil yang signifikan lebih baik ketimbang algoritma klasifikasi *machine learning* tradisional. Hasil pengujian juga menunjukkan bahwa proses analisis menggunakan pendekatan hibrid memiliki hasil sekitar 4-5% lebih akurat dalam mendeteksi *malware* android ketimbang hanya melakukan analisis statis atau analisis dinamis. Setelah melalui proses evaluasi dan penyesuaian *hyper* parameter hasil pengujian model menggunakan data uji menunjukkan akurasi dan skor F1 sebesar 98,7%. Hasil ini menunjukkan bahwa model sudah fit. Selama proses pengujian juga ditemukan fakta bahwa atribut yang paling membedakan antara aplikasi *malware* dan bukan *malware* adalah *read phone state*, di mana *read phone state* adalah fitur yang paling umum digunakan pada malware. Melalui fitur tersebut memungkinkan aplikasi mengetahui nomor telepon, informasi jaringan seluler yang *digunakan* dan status panggilan yang sedang berlangsung. Untuk penelitian ke depan perlu dipertimbangkan pengukuran terhadap penggunaan sumber serta waktu untuk deteksi *malware* mengingat pada perangkat *mobile* sumber daya lebih terbatas.

DAFTAR PUSTAKA

[1] S. Pichai, “Google’s I/O developer conference 2017”, [Online]. Available: https://events.google.com/io2017/. [Access on 10 July 2020].
[2] [Net Marketshare, “Operating System Market Share,” [Online]. Available: https://netmarketshare.com/operating-system-market-share.aspx. [Access on 10 July 2020].
[3] [P. Palumbo, L. Sayfullina, D. Komashinskiy, E. Eirola, and J. Karhumen, “A Pragmatic Android Malware Detection Procedure,” Computers & Security, July 2017
[4] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, and M. Conti, “Android Security: A Survey of Issues, Malware Penetration and Defenses,” IEEE Communications Surveys & Tutorials, Vol. 17, pp. 998 – 1022, 2015.
[5] F. I. Abro, “Investigating Android permissions and intents for malware detection,” Unpublished Doctoral thesis, University of London, 2018.
[6] K. Sugunan, T.G. Kumar, and K.A. Dhanya, “Static and Dynamic Analysis for Android Malware Detection,” Advances in Big Data and Cloud Computing, pp. 147 – 155, April 2018.
[7] R. B. Hadiprakoso, H. Kabetta, I. K. Buana “Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection” International Conference of Informatics, Multimedia, Cyber, and Information System (ICIMCIS), 2020
[8] R. B. Hadiprakoso, “Android Malware Dataset”, [Online]. Available: https://github.com/azureice10/malware /. [Access on 30 July 2021].
[9] F.Y. Osisanwo, J.E.T. Akinsola, O. Awodele, J.O. Hinmikaiye, O. Olakanmi, and J. Akinjobi, “Supervised Machine Learning Algorithms: Classification and Comparison”. International Journal of Computer Trend and Technology (IJCTT), Vol. 48, pp. 128 – 138, 2017.
[10] N. V. Duc, P. T. Giang dan P. Minh, “Permission Analysis for Android Malware Detection,” in The Proceedings of the 7th VAST, Hanoi, 2015.
[11] [S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, “StormDroid: A Streaminglized Machine Learning –Based System for Detecting Android Malware,” ASIA CCS ’16: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 377 – 388, May 2016.
[12] F. Idreesa, M. R. M. Contib, T. M.Chena dan Y. Rahulamathavan, “PIndroid: A novel Android malware detection system using ensemble learning methods,” *Computers & Security*, vol. 68, pp. 36-46, 2017
[13] [S. Malik, “Android System Call Analysis for Malicious Application Detection,” International Journal of Computer Sciences and Engineering, vol. 5, no. 11, pp. 105-108, 2017.
[14] Li, Xiang, et al. "An android malware detection method based on android manifest file." 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS). IEEE, 2016.
[15] Kabakus, Abdullah Talha, and Ibrahim Alper Dogru. "An in-depth analysis of Android malware using hybrid techniques." *Digital Investigation* 24 (2018): 25-33.
[16] Damodaran, Anusha, et al. "A comparison of static, dynamic, and hybrid analysis for malware detection." *Journal of Computer Virology and Hacking Techniques* 13.1 (2017): 1-12.
[17] Ali-Gombe, Aisha I., et al. "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming." *computers & security* 73 (2018): 235-248.
[18] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, “ANASTASIA: Android Malware detection using Static analysis of Applications,” 2016 8th IFIP Internasional Conference on New Technologies, Mobility and Security, November 2016.
[19] Z. Rehman, S.N. Khan, K. Muhammad, J.W. Lee, Z. Ly, S.W. Baik, P.A. Shah, K. Awan, and I. Mehmood, “Machine Learning-assisted Signature and Heuristic-based Detection of Malwares in Android Devices,” Computers & Electrical Engineering, Vol. 69, pp. 828 – 841, July 2018
[20] Surendran, Roopak, Tony Thomas, and Sabu Emmanuel. "A TAN based hybrid model for android malware detection." *Journal of Information Security and Applications* 54 (2020): 102483.
[21] Z. Yuan, Y. Lu, Y. Xue “Droiddetector: android malware characterization and detection using deep learning,” Tsinghua Sci. Technol., 21 (1) (2016), pp. 114-123.
[22] M. K. Alzaylaee, S.Y.Yerimab, S. Sezerc, “DL-Droid: Deep learning based android malware detection using real devices” Computers & Security Vol. 89, February 2020.