JURNAL ILMU TEKNIK DAN INFORMATIKA (TEKNIK) VOL 3, No.1, Mei 2023, pp. 19 - 34
p-ISSN : 2808-8751 (print)
e-ISSN : 2798-2513 (online)
http://journal.stiestekom.ac.id/index.php/TEKNIK

Deteksi Malware Statis Menggunakan Deep Neural Networks Pada Portable Executable

Dieta Wahyu Asry[1], Eko Siswanto[2], Dendy Kurniawan[3,] Haris Ihsanil Huda[4]
[1]Universitas Sains dan Teknologi Komputer Semarang
Jl. Majapahit 605 Semarang, e-mail: dieta@gmail.com
[2]Universitas Sains dan Teknologi Komputer Semarang
Jl. Majapahit 605 Semarang, e-mail: eko.siswanto@stekom.ac.id
[3]Universitas Sains dan Teknologi Komputer Semarang
Jl. Majapahit 605 Semarang, e-mail: dendy@stekom.ac.id
[4]Universitas Sains dan Teknologi Komputer Semarang
Jl. Majapahit 605 Semarang, e-mail: haris@stekom.ac.id

ARTICLE INFO

Article history:

Received Maret 2023
Received in revised form
Accepted April 2023
Available online Mei 2023

Abstrak

Latar Belakang: Dua komponen utama pada analisismalware adalah analisis malware statis yang melibatkan pemeriksaan struktur dasar malware yang dapat dieksekusi tanpa mengeksekusinya sedangkan analisis malware dinamis bergantung pada pemeriksaan perilaku malware setelah menjalankannya di lingkungan yang terkendali. Analisis malware statis biasanya dilakukan oleh perangkat lunak anti-malware modern dengan menggunakan analisis berbasis tanda tangan atau analisis berbasis heuristik. Tujuan Utama: Tujuan dari penelitian ini adalah megusulkan dan mengevaluasi deep neural network untuk menganalisis file portabel secara statis guna mempelajari fitur-fitur dari portable executable malware untuk meminimalkan terjadinya false positive saat mengenali malware baru. Metode penelitian: Model yang diusulkan dalam penelitian ini adalah model Neural Network with Dropout terhadap model pohon keputusan untuk memeriksa seberapa baik kinerjanya dalam mendeteksi file PE berbahaya yang sebenarnya. Metode format-agnostik digunakan untuk mengekstrak fitur dari file. Dataset digunakan untuk melatih model yang diusulkan dan membandingkan hasil dengan dataset malware lain yang diketahui. hasil: Hasil dari penelitian ini menunjukkan bahwa penggunaan jaringan saraf dalam yang sederhana untuk mempelajari fitur PE vektor tidak hanya efektif, tetapi juga kurang intensif sumber daya dibandingkan dengan metode deteksi heuristik konvensional. kesimpulan: Model yang diusulkan dalma penelitian ini mencapai AUC sebesar 99,8% dengan 98% true positive pada 1% false positive pada kurva ROC. Untuk menunjukkan bahwa model ini berpotensi melengkapi atau menggantikan perangkat lunak anti-malware konvensional maka

Received, 2023; Revised, 2023; Accepted, 2023

p-ISSN : 2808-8751 e-ISSN : 2798-2513

untuk penelitian dimasa depan diusulkan untuk mengimplementasikan model ini secara praktis.

Kata Kunci: *Malware detection, Deep Neural Network, Portable Executable*

.

1. PENDAHULUAN

Sejak munculnya perangkat lunak anti-malware, peningkatan malware canggih yang secara khusus dirancang untuk mengakali perangkat lunak ini hingga pada gilirannya mempelopori penelitian menjadi teknik deteksi yang lebih maju. Konsep deteksi malware berkaitan dengan menganalisis file yang dapat dieksekusi untuk menetapkan niat jahat. Analisis malware atau deteksi malware ini dapat dilakukan dengan dua cara: secara statis, dan dinamis. Deteksi malware statis (DMS) merupakan proses analisis file biner tanpa menjalankannya. Ini dapat melibatkan penyebaran file secara menyeluruh dan mampu memerikasa setiap komponen, menggunakan disassembler untuk merekayasa baliknya, atau mengubahnya menjadi kode rakitan untuk memeriksa alirannya Sikorski & Honig, (2012). Rekayasa tersebut masih dapat diperluas ke kode sumber asli perangkat lunak jika tersedia Manuel et al, (2012). DMS merupakan garis pertahanan pertama melawan malware yang digunakan oleh semua perangkat lunak anti-malware. Deteksi malware dinamis (DMD) menggunakan analisis perilaku saat malware berjalan untuk menentukan niat jahat, ini dilakukan di lingkungan sandbox untuk memastikan bahwa eksekusi tidak membahayakan sistem target. Bentuk analisis ini biasanya padat sumber daya dan dapat dielakkan dengan berbagai cara. Debugger juga dapat digunakan untuk menganalisis panggilan sistem atau pola perilaku lain yang tidak dapat dideteksi menggunakan pengujian kotak hitam Dilshan (2016).

Ruang lingkup penelitian ini hanya akan berfokus pada deteksi malware statis. Machine Learning (ML) telah lama digunakan untuk mengklasifikasikan data dengan karakteristik kompleks yang tidak dapat ditentukan dengan mudah menggunakan fungsi matematika. Deep Neural Network (DNN) saat ini digunakan dalam berbagai aplikasi yang berbeda termasuk klasifikasi data, prediksi data, pengenalan gambar, pemrosesan bahasa alami, dan sebagainya. Keserbagunaan jaringan saraf ini sangat cocok untuk big data di mana data dalam jumlah besar tersedia, tetapi secara komputasi proses untuk mendapat hasil tertentu memiliki harga yang sangat mahal. Hingga saat ini, kurangnya ketersediaan dataset berlabel untuk pembelajaran terawasi telah memperlambat kemajuan dalam penggunaan Machine Learning atau pembelajaran mendalam untuk deteksi malware. Igor Santos dkk. mengusulkan OPEM (Igor et al, 2013) sebagai pendekatan statis-dinamis untuk menggunakan Machine Learning untuk mendeteksi malware yang tidak dikenal. Mereka mengusulkan menganalisis kode operasional yang diperoleh dari pembongkaran executable dan menganalisis jejak eksekusi mereka untuk menentukan niat jahat. Demikian pula, kerangka kerja deteksi malware dinamis untuk Android bernama DroidDolphin berhasil mencapai akurasi 86,1% (Wu & Hung, 2014) menggunakan analisis malware dinamis. Kedua metode umumnya mahal secara komputasi dan memiliki keterbatasan ketersediaan data berlabel.

Tujuan utama dari penelitian ini adalah untuk merancang dan mengevaluasi deep neural network untuk menganalisis secara statis file portabel yang dapat dieksekusi untuk mengklasifikasikannya sebagai berbahaya atau jinak. Karena tujuan tersebut, penelitian ini menggunakan dataset yang berisi data yang diekstrak dari file portabel yang dapat dijalankan yang berbahaya dan tidak berbahaya. Trik yang digunakan untuk ekstraksi file portabel ini adalah hashing (Smola, 2009) untuk membuat ringkasan matematis dari fitur dan menstandarkan vektor input. Selanjutnya model penelitian ini dibandingkan dengan model serupa yang diusulkan sebelumnya untuk menangani analisis statis malware. Penelitian ini menunjukkan model yang mensimulasikan implementasi praktis dari model yang serupa untuk penelitian lebih lanjut. Sumber lengkap untuk mereproduksi model yang diusulkan dan hasil turunannya disediakan dalam penelitian ini. terbit di jurnal ini tanpa melalui proses lebih lanjut untuk melengkapi tulisan dan data hasil akhir penelitian.

2. TINJAUAN PUSTAKA

Ada berbagai tantangan yang terkait dengan analisis malware statis. Sebagian besar masalah ini dapat diselesaikan dengan menggunakan analisis malware dinamis seperti kerusakan file selama runtime, pengaburan kode, atau file biner terenkripsi yang dapat dijalankan. Dallam literatur review ini beberapa masalah dan kekurangan penganalisa semantik dieksplorasi. Biasanya, perangkat lunak anti-virus menggunakan metode berbasis tanda tangan untuk mendeteksi malware. Instruksi yang ada di malware

p-ISSN : 2808-8751 e-ISSN : 2798-2513

executable diuraikan untuk mendapatkan tanda tangan unik yang mengidentifikasi malware yang kemudian dibandingkan dengan database besar tanda tangan malware yang dikenal (Ammar et al, 2012; Philip et al, 2011). Bonfante et al, (2007) mengusulkan metode grafik aliran kontrol untuk mengatasi masalah ini. Mereka menggunakan grafik dengan node untuk semua instruksi perakitan yang umum digunakan, lalu menggunakan versi yang diperkecil dari grafik ini sebagai tanda tangan untuk mengklasifikasikan malware. Menurut pengujian mereka, bentuk deteksi ini menghasilkan akurasi deteksi keseluruhan yang lebih baik saat grafiknya lebih besar (untuk executable yang lebih besar). Analisis malware statis terutama telah dipelajari dari perspektif analisis semantik dan analisis kode sumber untuk klasifikasi. Moser dkk. mengusulkan metode untuk mengaburkan kode dari analisis semantik (Andreas et al, 2007) hanya dengan menggunakan konstanta buram untuk mengaburkan aliran kontrol program. Ini berfungsi untuk menyoroti kelemahan yang signifikan dalam teknik analisis malware statis yang ada saat ini dengan analisis semantikaware di mana analisis semantik dapat dikalahkan dengan memperkenalkan pendekatan acak untuk menghitung konstanta secara real time. Salah satu metode yang disebutkan adalah menggunakan benih acak untuk menghasilkan alamat tempat variabel disimpan, atau untuk menghubungkan proses dan menyimpan variabel di alamat yang ada di alamat lain.

Kebingungan kode menggunakan enkripsi pada file biner beberapa kali dan kemudian menggabungkan alat untuk dekripsi dibahas oleh Christodorescu and Jha (2006). Bentuk kebingungan ini mudah ditangkap selama runtime dengan menganalisis file yang didekripsi dalam memori, tetapi sulit untuk menentukan tingkat enkripsi file tanpa dekripsi terlebih dahulu dan menganalisisnya secara dinamis. Pendekatan berbasis semantik yang mengusulkan metrik untuk mengukur kesamaan antara kode malware asli dan kode malware yang disamarkan diusulkan oleh Preda et al, (2007). Ada banyak metode deteksi malware dinamis termasuk analisis grafik panggilan (Ammar et al, 2012) dan mengidentifikasi perilaku berdasarkan pemicu (Brumley et al, 2008). Namun, secara komputasi metode ini mahal dan membutuhkan infrastruktur sandbox tempat malware dapat dieksekusi dan dianalisis dengan aman. Pengepakan file adalah teknik umum yang digunakan saat membundel perangkat lunak besar dalam paket yang kecil dan ringkas (Philip et al, 2011). Teknik pengemasan seperti itu biasanya melibatkan beberapa bentuk enkripsi yang berpotensi mencegah identifikasi malware dengan mudah. Salah satu alat yang disebut PolyPack (Oberheide et al, 2009) secara khusus dirancang untuk membuktikan bahwa pengemas adalah metode yang efektif untuk menghindari perangkat lunak anti-virus dan anti-malware. Mereka menyediakan 10 pengemas mengemas data yang diberikan secara mandiri, kemudian memindai data yang dikemas dengan 10 pemindai anti-virus terkenal, lalu pengemas dengan hasil terbaik dipilih. Studi mereka menetapkan bahwa ini meningkatkan tingkat penghindaran sebesar 2,58 kali terhadap sebagian besar perangkat lunak anti-virus.

Fakta bahwa Machine Learning bekerja lebih baik dengan dataset yang lebih besar sudah mapan (Banko and Brill, 2001). Beberapa penelitian telah dipublikasikan yang menggunakan Machine Learning untuk klasifikasi malware. Berbagai metode seperti analisis dinamis panggilan sistem (Kolosnjaji et al, 2016), pemantauan akses registri (Heller et al, 2003) analisis berbasis model Markov tersembunyi (Attaluri et al, 2009) telah diusulkan untuk analisis malware dinamis. Kolter dan Maloof (2004) mengusulkan penggunaan n-gram dengan menggabungkan urutan 4 byte untuk menghasilkan sekitar 255 juta n-gram yang berbeda. Penelitian ini berfokus pada masalah klasifikasi malware daripada masalah pendeteksian malware. Penelitian ini mengusulkan penggunaan pendekatan probabilistik untuk menentukan fitur mana yang relevan dan menggunakan 500 n-gram teratas untuk analisis. Makalah ini mengusulkan penggunaan Naive Bayes, Support Vector Machine (SVM), dan pohon keputusan J48 untuk menganalisis data mereka. Data yang digunakan untuk analisis sebagian besar bersumber dari Sourceforge dan VX Heavens (data aktual tidak diungkapkan) dengan 1971 file executable jinak dan 1651 file executable berbahaya diuji. Kumpulan sampel kecil yang digunakan dalam penelitian ini, dan fakta bahwa dataset yang tepat yang digunakan tidak tersedia, sulit untuk memastikan keakuratan hasil ini bila digunakan dengan dataset yang lebih besar. Studi serupa dilakukan oleh Bagga (2017) menggunakan pendekatan ini dengan Microsoft Malware Classi cation yang merupakan dataset yang bisa dibilang besar.

Raman, dari Tim Respons Insiden Produk di Adobe Systems Inc. mengusulkan metode untuk mengklasifikasikan malware dengan mengekstraksi tujuh fitur yang paling tidak berkorelasi dari executable portabel (Raman et al, 2012). Fitur yang diekstraksi adalah DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, Virtu-alSize, NumberOfSections. Dataset yang berisi 100.000 file executable berbahaya dan 16.000 file executable jinak digunakan untuk eksperimen. Berbagai model diuji menggunakan data ini. Di antara model yang diuji, pohon keputusan J48 (Quinlan, 1993) memperoleh hasil terbaik: tingkat positif

p-ISSN : 2808-8751 e-ISSN : 2798-2513

sejati 0,986 dengan tingkat positif palsu 0,057. Model terlatih yang dihasilkan dirilis sebagai alat gratis untuk klasifikasi malware, tetapi dataset tidak dipublikasikan untuk melakukan segala bentuk penelitian komparatif. Anderson dan Roth selanjutnya menguji model terlatih ini dengan dataset dan menemukan bahwa model tersebut menunjukkan tingkat positif palsu 0,53 dan tingkat negatif palsu 0,08. Model klasifikasi malware dinamis menggunakan jaringan saraf dalam yang disebut MtNet diusulkan oleh Huang dan Stokes pada tahun 2016 (Huang and Stokes, 2016). Dataset yang digunakan untuk penelitian ini disediakan oleh Microsoft Corporation yang berisi 6,5 juta file sampel. Dari dataset ini, 2,85 juta file berbahaya dan 3,65 juta file jinak diekstrak. Fitur pelatihan diekstraksi selama eksekusi file saat runtime terutama terdiri dari dua jenis data: pemanggilan fungsi sistem dan objek yang dihentikan null. Seleksi fitur dilakukan dengan menggunakan mutual information yang diusulkan oleh Manning et al. (2010) untuk mendapatkan total 50.000 fitur masukan. Tujuan akhirnya adalah untuk mengklasifikasikan malware terlebih dahulu sebagai jinak atau berbahaya, lalu mengklasifikasikan malware jahat ke dalam salah satu dari 100 keluarga malware yang dikenal. Fungsi aktivasi ReLU digunakan bersama dengan lapisan dropout yang ditambahkan untuk kinerja model yang lebih baik. Meskipun model ini menunjukkan hasil yang mengesankan dengan tingkat positif palsu di bawah 0,07%, kurangnya ketersediaan dataset pengujian dan kode model yang digunakan untuk pengujian membuat reproduksi hasil ini tidak mungkin dilakukan. Jaringan status gema dan pengklasifikasi malware berbasis jaringan saraf berulang juga telah diuji untuk analisis dinamis malware oleh Pascanu et al, (2015). Penelitian mereka menetapkan penggunaan model berulang berbasis jaringan keadaan gema dengan fungsi aktivasi sigmoid (regresi logistik) untuk analisis dinamis malware. Vektor input yang tepat tidak diungkapkan, namun berasal dari panggilan API yang dilakukan oleh file selama eksekusi runtime. Model mencapai tingkat positif sejati 0,983 dengan tingkat positif palsu 0,001. Dataset yang digunakan dalam penelitian ini bersumber secara internal, dan tidak tersedia untuk umum. Tujuan dari penelitian ini adalah untuk menetapkan bahwa jaringan saraf berulang dapat digunakan untuk analisis malware dinamis. Namun, karena dataset tidak tersedia, dan langkah-langkah yang diperlukan untuk mereproduksi model yang diusulkan tidak tersedia, sulit untuk memverifikasi hasil ini dan melakukan penelitian lebih lanjut didasarkan dari itu.

Seleksi Fitur

Machine Learning sangat sensitif terhadap rangkaian fitur yang digunakan untuk pelatihan. Berbagai penelitian telah menetapkan fitur-fitur tertentu yang bermanfaat untuk pelatihan yang efektif dari pengklasifikasi malware berbasis Machine Learning. Divandari dkk. mengusulkan penggalian data opcode dari file dan menggunakan pendekatan Markov Blanket untuk meringkas set fitur (Divandari et al, 2015). Karena opcode sendiri merupakan bagian yang signifikan dari executable, mereka telah dianggap sebagai fitur yang dapat diandalkan untuk deteksi malware (Bilar, 2007). Model yang diusulkan menggunakan Hidden Markov Model (HMM) untuk klasifikasi malware. Pendekatan histogram byte yang diusulkan oleh Saxe dan Berlin dalam penelitian mereka (Saxe and Berlin, 2015) memperkenalkan metode format-agnostik untuk mengekstrak fitur dari file. Metode ini merupakan pendekatan inovatif untuk mengekstraksi informasi byte sebagai fitur dari file tanpa memerlukan informasi tentang fungsi sebenarnya dari byte tersebut dan menyarankan untuk mengekstrak histogram dari semua nilai byte yang ada dalam file biner bersama dengan histogram byte-entropi 2 dimensi untuk membangun pemahaman tentang potensi enkripsi atau kompresi yang digunakan dalam file.

Dalam model penelitian ini, metode format-agnostik untuk mengekstrak fitur dari file digunakan untuk melengkapi metode ekstraksi tajuk sedemikian rupa sehingga dapat mencapai akurasi tinggi tanpa overhead tinggi yang diperlukan untuk membuat vektor semua byte dalam executable portabel. Trik hashing fitur yang diusulkan oleh Weinberger et al. (2009) telah sering dikutip dan digunakan untuk model Machine Learning. Vektor input untuk sebagian besar model berbasis Machine Learning bersifat statis dan tidak dapat ditingkatkan ukurannya tergantung pada ukuran input. Oleh karena itu, diperlukan suatu metode untuk secara efektif meringkas fitur input besar ke dalam ukuran statis yang lebih mudah dikelola untuk pelatihan. Trik hashing fitur mengusulkan metode untuk secara efektif mengurangi dimensi data sehingga masih cukup mewakili data asli yang dimaksudkan, tetapi menawarkan fitur yang dapat dipisahkan secara linier untuk melatih model secara efektif.

Pohon Keputusan yang Didorong Artificial Neural Networks

p-ISSN : 2808-8751 e-ISSN : 2798-2513

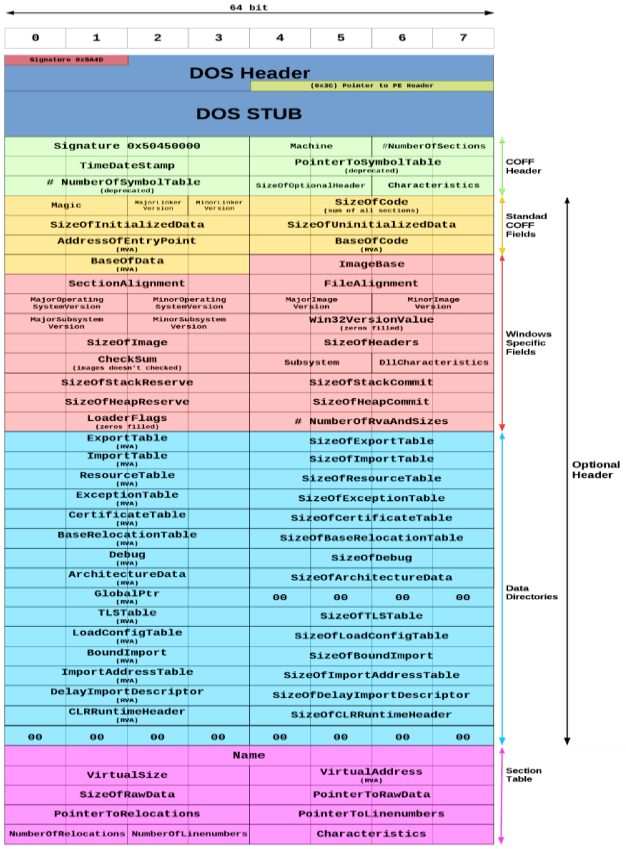
Metode boosting untuk model pohon keputusan telah terbukti memiliki kinerja yang sama atau lebih baik daripada jaringan saraf tiruan. Metode tersebut relatif lebih mudah untuk disetel dan bekerja dengan baik dengan sejumlah besar variabel (Roe et al, 2005). Dengan munculnya AdaBoost, peningkatan model pohon keputusan telah berhasil beralih dari klasifikasi biner ke klasifikasi multi-kategori (Hastie et al, 2009; Schapire, 2003). Ini mendorong penggunaan model berbasis pohon keputusan yang dikuatkan sebagai alternatif untuk jaringan saraf tiruan. Model yang diusulkan dalam penelitian ini dibandingkan dengan model pohon keputusan terdorong yang sudah ada untuk dataset yang sama digunakan untuk model yang diusulkan dalam penelitian ini. Caruana dan Niculescu-Mizil menetapkan dalam makalah mereka (Cruana and Mizil, 2006) bahwa menyatakan mesin vektor, pohon keputusan yang ditingkatkan, dan jaring saraf memiliki kinerja yang sebanding di sebagian besar skenario dengan varians terutama terbatas pada penyetelan hiper-parameter

Format Portabel yang Dapat Dieksekusi

Format portable executable (PE) diperkenalkan oleh Microsoft dengan sistem operasi Windows NT 3.1. Sejak awal, telah terlihat beberapa peningkatan untuk memasukkannya ke dalam versi Windows yang lebih baru. Unix menggunakan format ELF yang analog dengan format Windows PE. Cakupan tesis ini terbatas pada executable Windows karena data yang tersedia untuk malware yang berjalan pada sistem operasi berbasis Unix terbatas. Namun, header COFF ditemukan di PE file umum untuk lingkungan Unix dan Windows (Kath, 1993). Model yang diusulkan dalam penelitian ini menganalisis fitur yang diekstraksi dari file PE untuk menentukan apakah file tersebut berbahaya atau jinak. Bagian ini menjelaskan informasi yang dapat diperoleh dari file PE.

MS-DOS

Rintisan ini dijalankan setiap kali file dijalankan di lingkungan MS-DOS. Tujuan satu-satunya adalah mencetak pesan yang menunjukkan bahwa file tidak dapat dijalankan di lingkungan MS-DOS. Tanda tangan yang ditambahkan setelah rintisan MS-DOS menunjukkan bahwa file tersebut dalam format PE.



p-ISSN : 2808-8751 e-ISSN : 2798-2513

Gambar 11: Format File PE (Wikimedia Commons, 2016)

Dataset

Karena model yang diusulkan disini menganalisis file PE, tantangan pertama adalah menemukan dataset yang menyediakan file PE berlabel berbahaya atau jinak. Sulit untuk menemukan set data yang mengklasifikasikan malware sebagai berbahaya atau jinak. Dataset ini berukuran 9,1 GB dan menyediakan 900 ribu sampel pelatihan dengan 300 ribu sampel berbahaya, 300 ribu jinak, dan 300 ribu sampel tidak berlabel. Ini juga berisi 200 ribu sampel uji. Penerbit dataset ini juga telah merilis kode sumber yang mereka gunakan untuk membuat dataset ini. Kode ini digunakan sebagai dasar untuk mengekstrak fitur dari file biner. Data disediakan dalam file JSON di mana setiap baris adalah satu sampel. Setiap sampel berisi fitur yang diurai, yang pada dasarnya adalah fitur yang diekstraksi dari header PE dan memformat fitur agnostik.

Histogram byte

Histogram byte pada dasarnya adalah vektor berukuran 256 dengan setiap indeks mewakili frekuensi dari nilai byte yang sesuai. Misalnya, jika ada 11203 kemunculan nilai byte 200, maka nilai yang ada pada indeks 200 vektor histogram adalah 11203.

Histogram Entropi Byte

Untuk menghitung histogram entropi byte, jendela berukuran 2048 byte dipindahkan ke byte masukan dengan ukuran langkah 1024 byte. Entropi seluruh jendela 2048 byte dengan kemunculan setiap byte individu di jendela dihitung dan disimpan sebagai pasangan. Ini menghasilkan total 2048 pasang. Bin ukuran 16 x 16 digunakan untuk mengkuantisasi nilai entropi dan byte (Saxe and Berlin, 2015), AR18]. Nilai-nilai ini dinormalisasi sebelum pelatihan.

MODEL YANG DIUSULKAN

Model memiliki 3 komponen utama yaitu ekstraksi Fitur dan Hashing, Penskalaan dan Normalisasi, dan Pengklasifikasi Neural Network. Untuk mengimplementasikan model penelitian ini menggunakan Python karena kesederhanaan dan fleksibilitas penggunaannya dalam aplikasi Machine Learning (Oliphant, 2007). Arsitektur Nvidia CUDA (Nickolls, 2008) juga dimanfaatkan untuk komputasi paralel berkecepatan tinggi menggunakan pustaka Keras (Collet et al, 2015) untuk mengimplementasikan model jaringan saraf yang diusulkan.

Ekstraksi Fitur dan Hashing

Ada dua komponen utama file PE yang diekstrak untuk melatih model Informasi yang Diurai dan Informasi Byte Mentah. Dataset yang digunakan dalam model penelitian ini menyediakan modul yang nyaman untuk mengekstrak data yang diperlukan dari file PE yang diberikan. Secara total, model yang digunakan adalah 2351 vektor masukan untuk klasifikasi.

Informasi yang Diurai

Setiap file PE berisi informasi header, fitur-fitur ini diekstraksi dengan python menggunakan pustaka LIEF untuk mem-parsing file PE (Thomas, 2017). Namun, informasi ini tidak semuanya numerik dan tidak selalu berukuran sama. Model penelitian ini menggunakan vektor input dengan ukuran tetap untuk pelatihan. Ini berarti bahwa semua fitur yang diekstraksi dari file PE harus dibawa ke ukuran standar sebelum dapat digunakan untuk pelatihan model. Untuk mencapai ini, modul FeatureHasher dari perpustakaan scikit-learn (Pedregosa, 2011) juga diguanakan untuk mengimplementasikan trik hashing fitur (Weinberger et al, 2009) dengan sejumlah bin per fitur header. Lima kelompok fitur diekstraksi dari file PE.

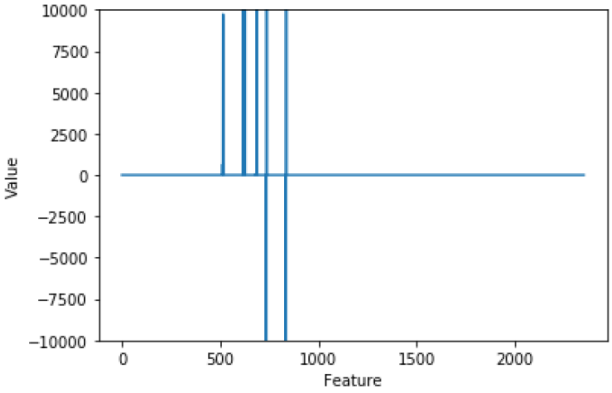
Informasi Umum

Fitur umum yang didapat dari file PE yaitu: Ukuran maya, Fungsi yang diimpor, Fungsi yang diekspor, Kehadiran bagian debug, Sumber daya, Relokasi, dan Jumlah Simbol

Penskalaan dan Normalisasi

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Setelah mengekstraksi fitur dari file PE diperoleh 2351 fitur per sampel. Namun, nilai fitur ini tersebar luas dengan sejumlah besar nilai mendekati 0 dan beberapa di antaranya di atas 109 dan beberapa di antaranya di bawah -104. Fenomena ini ditunjukkan sebagai grafik garis untuk salah satu sampel uji pada Gambar 5. Plot pencar dari fenomena ini mempersulit untuk melihat titik-titik ekstrem yang membuatnya perlu untuk memplot grafik garis.



Gambar 5: Grafik garis sampel mentah

Saat mencoba melatih model dengan jenis data ini, disini model tidak dapat menyatu dan menghasilkan AUC 0:5. Ini jelas karena skala fitur yang diekstraksi sangat bervariasi yang memerlukan beberapa bentuk normalisasi sebelum digunakan dalam model. Penelitian ini menggunakan normalisasi statistik atau Z-score untuk tujuan ini (Jayalakshmi and Santhakumaran, 2011). Diketahui sampel input menjadi X = {X_1; X_2; X_3… X_{2351}} di mana X_i mewakili fitur ke-i dalam sampel. Normalisasi statistik sampel X untuk setiap fitur Xi dilakukan dengan menggunakan persamaan:

$$X_I' = \frac{X_i - X}{\sigma(X)}$$

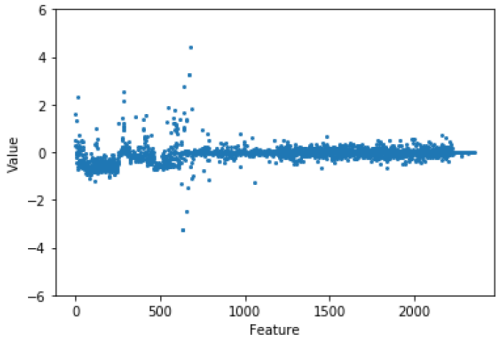
Dimana X adalah rata-rata aritmetika dari sampel X dan σ(X) adalah standar deviasi dari sampel X. Rata-rata aritmatika X dari $\mathcal{X}$dihitung menggunakan persamaan:

$$\mathcal{X} = \frac{1}{2351}\sum_{I=1}^{1} X_I$$

Dimana 2351 adalah kardinalitas sampel X. Standar deviasi σ(X) dari X dihitung dengan menggunakan persamaan:

$$\sigma(X) = \sqrt{\frac{1}{2351}\sum_{I=1}^{2351}(X_I - \mathcal{X})^2}$$

Disini menggunakan fungsi **StandardScalar** yang disediakan di perpustakaan **scikit-learn** (Pedregosa et al, 2011) untuk menormalkan sampel yang digunakan. Fungsi ini melakukan perhitungan yang sama persis.



p-ISSN : 2808-8751 e-ISSN : 2798-2513

Gambar 6: Scatter plot dari sampel yang dinormalisasi

Pengklasifikasi Neural network

Penelitian ini menggunakan untuk menganalisis data. Ada dua jaringan saraf yang dibangun, satu dengan lapisan dropout, dan satu lagi tanpa lapisan dropout. Fungsi aktivasi logistik dan fungsi aktivasi recti ed linear unit (ReLU) diuji untuk kedua jaringan ini. Pengoptimal Adam and Kingma (2014) yang diimplementasikan di perpustakaan Keras digunakan untuk pengoptimalan berbasis gradien dari clasifier. Ringkasan dari kedua jaringan saraf ditunjukkan pada Gambar 7 dan Gambar 8. Jaringan tanpa lapisan dropout berisi 1 lapisan input yang menerima vektor input ukuran 2351, 1 lapisan padat dengan 2400 neuron, 3 lapisan padat dengan 1200 neuron, dan lapisan output biner. Jaringan kedua dengan 2 lapisan dropout dan 2 lapisan padat diusulkan untuk menguji potensi peningkatan kinerja yang terukur saat mengurangi jumlah lapisan dan memperkenalkan dropout.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2400)	5644800
dense_2 (Dense)	(None, 1200)	2881200
dense_3 (Dense)	(None, 1200)	1441200
dense_4 (Dense)	(None, 1200)	1441200
dense_5 (Dense)	(None, 1)	1201

Total params: 11,409,601
Trainable params: 11,409,601
Non-trainable params: 0

Gambar 7: Ringkasan jaringan saraf

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2400)	5644800
dropout_1 (Dropout)	(None, 2400)	0
dense_2 (Dense)	(None, 1200)	2881200
dropout_2 (Dropout)	(None, 1200)	0
dense_3 (Dense)	(None, 1200)	1441200
dense_4 (Dense)	(None, 1)	1201

Total params: 9,968,401
Trainable params: 9,968,401
Non-trainable params: 0

Gambar 8: Ringkasan jaringan saraf dengan lapisan dropout.

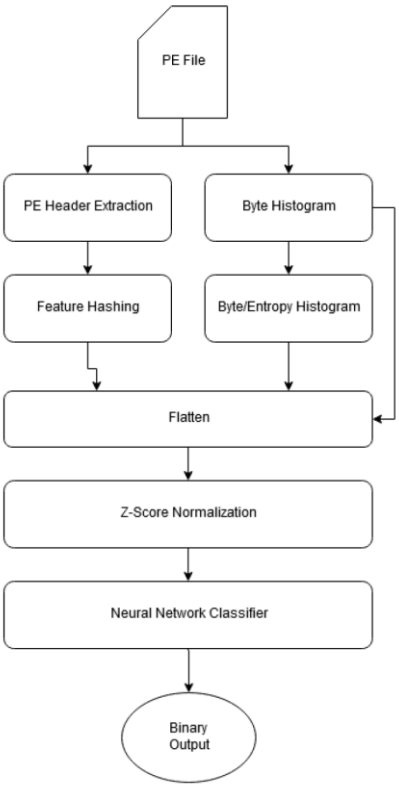
Ringkasan Model

Diagram ringkasan model ditunjukkan pada Gambar 9. Keseluruhan model terdiri dari:

1. Ekstrak fitur agnostik header dan platform dari file PE.
2. Gunakan trik hashing (Weinberger et al, 2009) untuk meringkas fitur header.
3. Ratakan fitur menjadi vektor input satu dimensi dengan ukuran 2351.
4. Normalisasi fitur menggunakan normalisasi statistik atau z-score.
5. Beri makan vektor input melalui jaringan saraf dalam yang terhubung erat untuk mendapatkan output biner.

Output akhir dari model ini adalah 0 atau 1. Di mana 0 menunjukkan bahwa file PE yang diuji tidak berbahaya, dan 1 menunjukkan bahwa file berbahaya.

p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 9: Diagram alir model yang diusulkan

3. HASIL DAN PEMBAHASAN

Metrik untuk Pengujian Model

Sebelum menguji model, penting untuk mengidentifikasi metrik yang akan digunakan untuk tujuan ini. Dalam kasus ini, penelitian ini menguji keakuratan dan kemampuan diagnostik model yang ada dalam penelitian ini, kurva karakteristik output penerima (ROC) diperoleh dan area di bawah kurva (AUC) ditemukan. Metode ini umumnya memberikan ukuran yang lebih baik dari kemampuan diagnostik classier dibandingkan dengan hanya menyatakan akurasi keseluruhan model terhadap set tes yang diberikan (Fawcett, 2006:861; Metz, 1978:283). Matriks kebingungan dari model berbasis jaringan saraf dan model berbasis pohon keputusan juga ditemukan terlebih dahulu untuk memudahkan menemukan kasus kesalahan klasifikasi yang tidak terlihat jelas dari kurva ROC. Kurva ROC diturunkan dengan memplot true positive rate (TPR) dari pengklasifikasi terhadap false positive rate (FPR). TPR dan FPR dihitung menggunakan persamaan berikut:

4. $$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$
5. $$FPR = \frac{FP}{P} = \frac{FP}{FP + TN}$$

Di mana TP adalah positif sejati, P adalah total sampel positif yang ada dalam set pengujian, dan FN adalah negatif palsu. FP adalah positif palsu, N adalah total sampel negatif yang ada dalam set tes, dan TN adalah negatif sebenarnya. **roc_curve, auc** digunakan dan modul dari **scikit-learn.metrics** untuk mendapatkan kurva ROC, dan modul **fusion_matrix** dari perpustakaan yang sama untuk mendapatkan matriks kebingungan. Semua data diplot menggunakan MatPlotLib. Sampel uji 200K yang disediakan dalam dataset digunakan untuk menguji model yang digunakan kemudian hasilnya dibandingkan dengan model berbasis pohon keputusan smenggunakan LightGBM (Guolin et al, 2017).

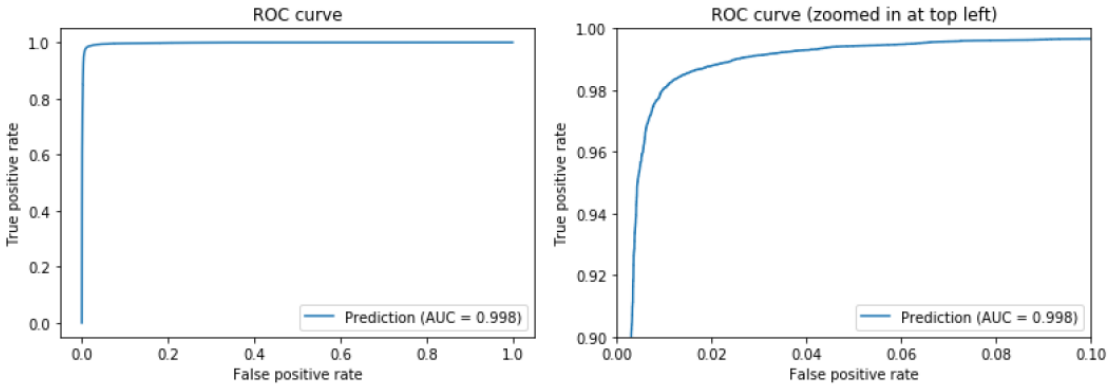
p-ISSN : 2808-8751 e-ISSN : 2798-2513

Hasil tes

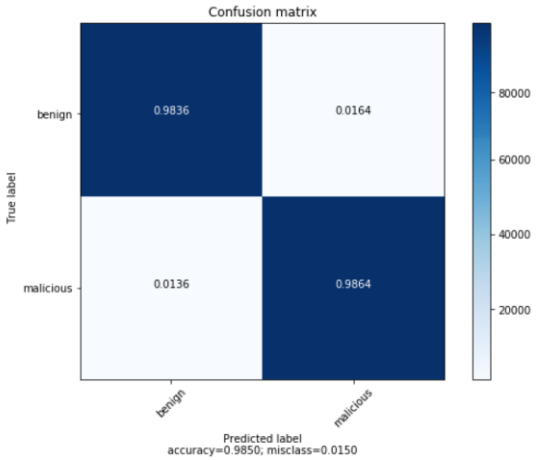
Classifier Type	AUC	TPR @ FPR = 0.01
Neural Network (Sigmoid)	0.998	0.981
Neural Network with Dropout (Sigmoid)	0.998	0.978
Neural Network (ReLU)	0.997	0.982
Neural Network with Dropout (ReLU)	0.997	0.989
Decision Tree using LightGBM	0.999	0.982

Tabel 6: Ringkasan hasil untuk pengklasifikasi berbasis jaringan saraf, dan untuk pengklasifikasi berbasis pohon keputusan

Untuk menguji model penelitian ini menggunakan 4 pengklasifikasi berbasis jaringan saraf yang berbeda, dan kemudian mengujinya menggunakan pengklasifikasi berbasis pohon keputusan. Ringkasan hasil dapat dilihat pada tabel 6, seperti yang terlihat kurva ROC pada Gambar 10, 12, 13, dan 15, AUC jaringan saraf menggunakan fungsi aktivasi ReLU sedikit lebih rendah daripada yang menggunakan fungsi aktivasi sigmoid. Selain itu, meskipun AUC dari pengklasifikasi pohon keputusan adalah yang tertinggi, tingkat positif sebenarnya dari model ini sama atau lebih rendah bila dibatasi pada tingkat positif palsu 1% dibandingkan dengan jaringan saraf berbasis ReLU. Matriks konfusi untuk model pada Gambar 11, 13, 15, 17, 19 menunjukkan visualisasi kinerja klasifikasi yang lebih baik dari masing-masing model. Karena ReLU tampaknya memperoleh hasil terbaik, kurva ROC dari jaringan saraf berbasis ReLU dibandingkan, dan pengklasifikasi pohon keputusan pada Gambar 19 dan 20.



Gambar 10: Model ROC Curve menggunakan Neural Network (Sigmoid)

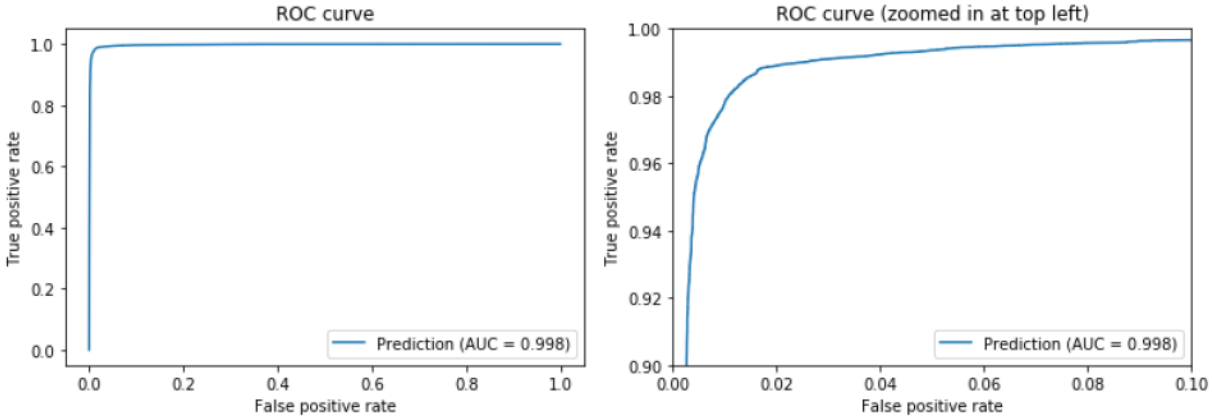


Gambar 11: Model Confusion Matrix menggunakan Neural Network (Sigmoid)

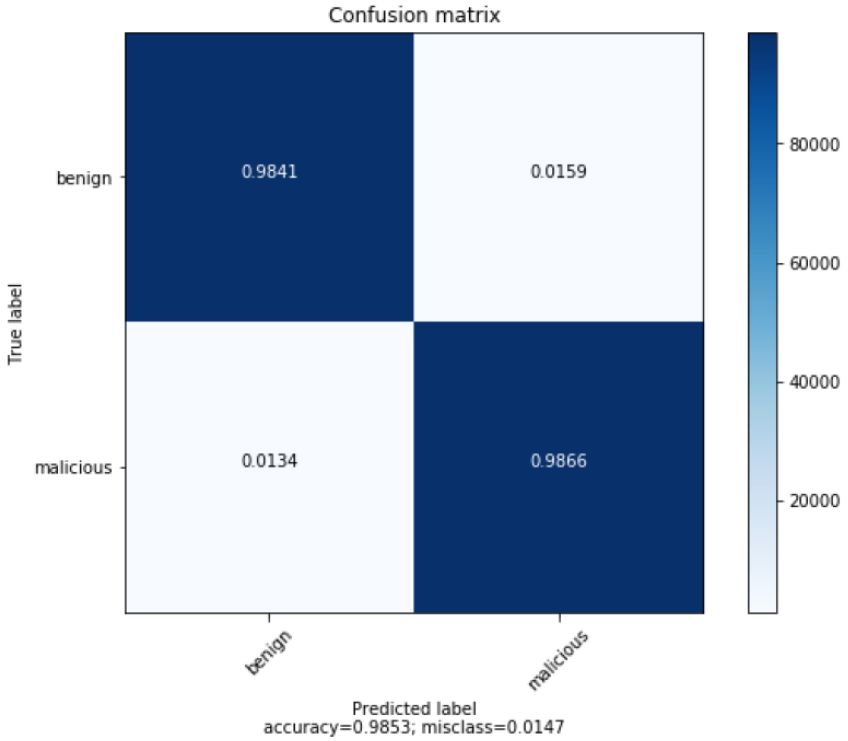
Pengujian Dunia Nyata

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Selanjutnya, penelitian ini menguji model dengan kinerja terbaik dari model yang diusulkan (Neural Network with Dropout (RelU)) terhadap model pohon keputusan untuk memeriksa seberapa baik kinerjanya dalam mendeteksi file PE berbahaya yang sebenarnya dan hasilnya dirangkum dalam Tabel 7. Penelitian ini menggunakan kumpulan sampel 997 sampel dari VirusShare.com (Robert, 2011) untuk pengujian. Sayangnya, sampel di VirusShare.com hanya dapat diakses dengan registrasi, sehingga sampel itu sendiri tidak dapat dibagikan secara publik. Nama file dari kumpulan sampel adalah: **VirusShare_x86-64_WinEXE_20130711.zip** Hasil yang diperoleh dari model berbasis jaringan saraf seperti yang diharapkan berdasarkan hasil yang sebelumnya. Dari 997 sampel yang diuji, 994 diklasifikasikan dengan benar. Namun model berbasis pohon keputusan hanya mampu mengklasifikasikan 117 dari 997 sampel. Sulit untuk menentukan penyebab dari hasil ini tanpa pengujian lebih lanjut.

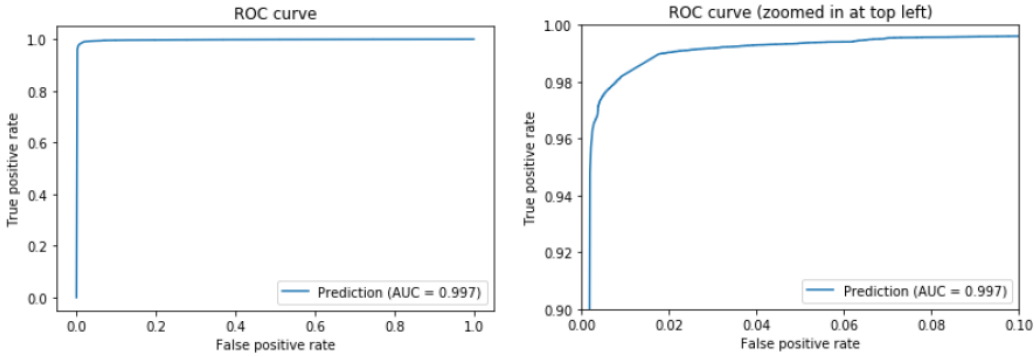


Gambar 12 Kurva ROC model menggunakan Neural Network dengan Dropout (Sigmoid)

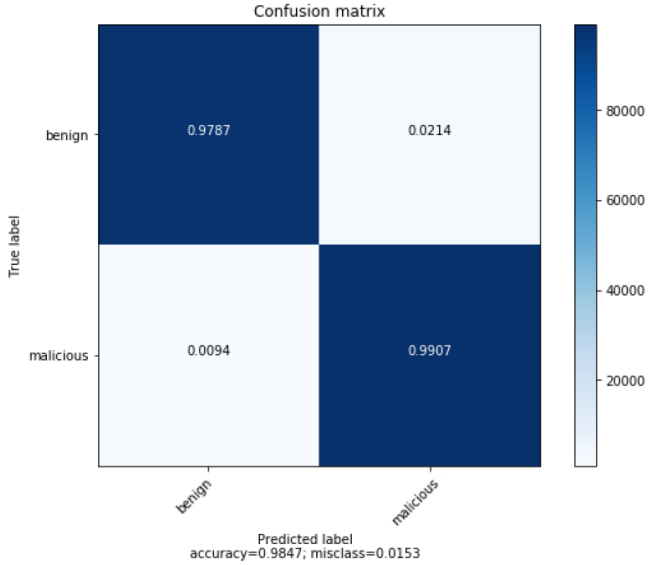


Gambar 13 Model Confusion Matrix menggunakan Neural Network dengan Dropout (Sigmoid)

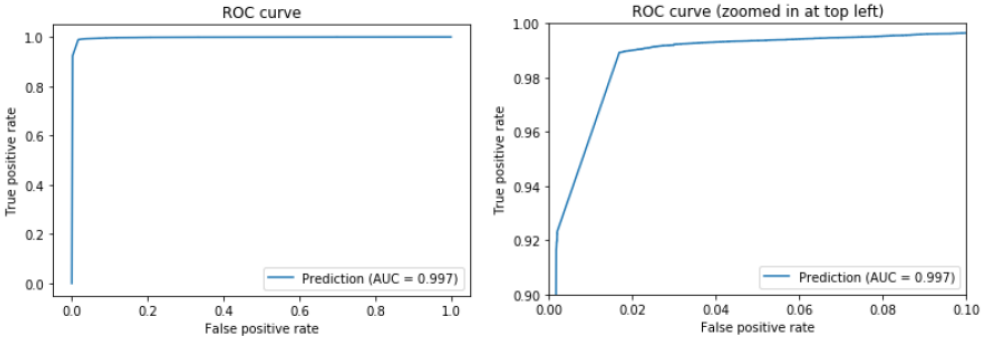
p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 14 Model ROC Curve menggunakan Neural Network (ReLU)

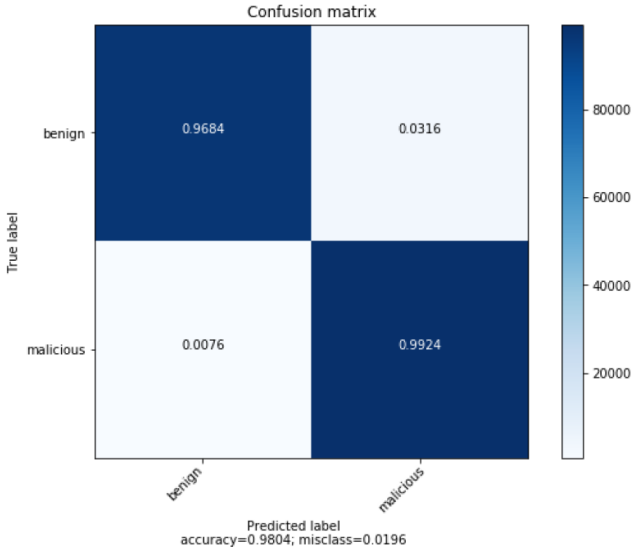


Gambar 15 Model Confusion Matrix menggunakan Neural Network (ReLU)

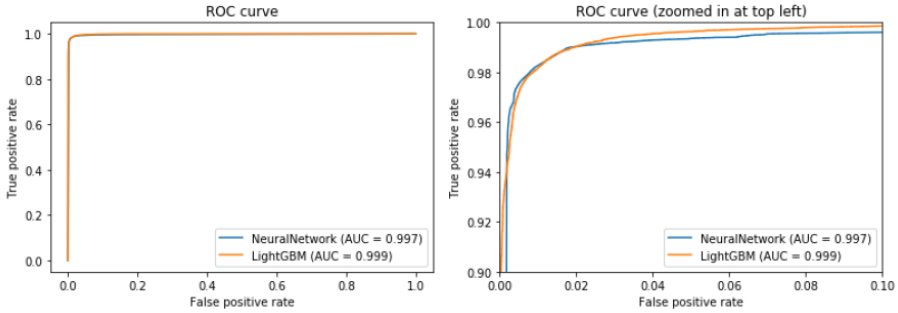


Gambar 16 Model ROC Curve menggunakan Neural Network with Dropout (ReLU)

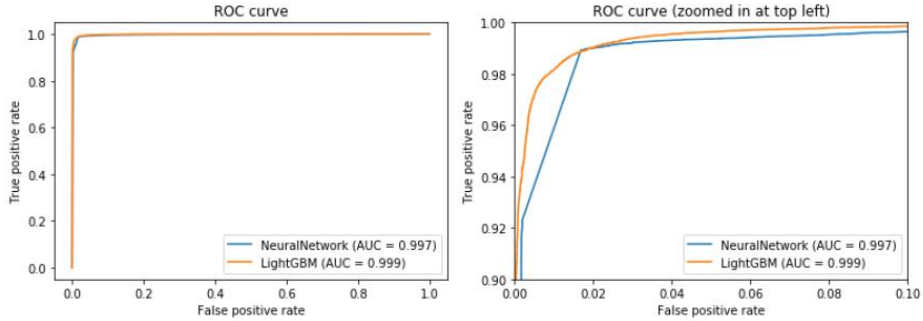
p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 17 Model Confusion Matrix menggunakan Neural Network with Dropout (ReLU)

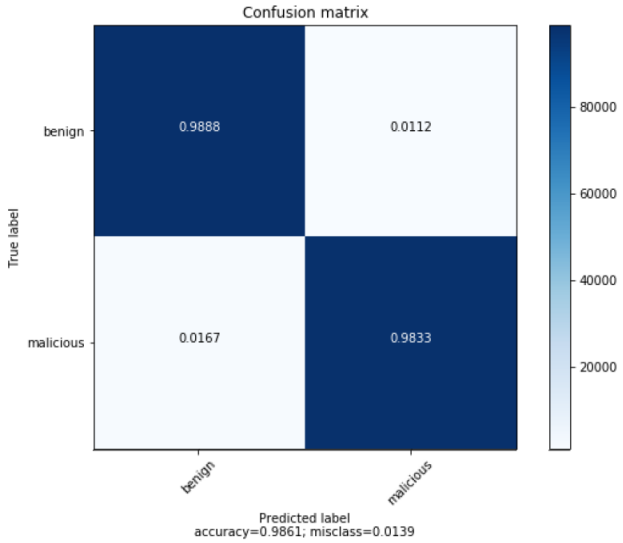


Gambar 18 Model Kurva ROC menggunakan Neural Network (ReLU) dan menggunakan Decision Tree



Gambar 19 Model ROC Curves menggunakan Neural Network with Dropout (ReLU) dan menggunakan Decision Tree

p-ISSN : 2808-8751 e-ISSN : 2798-2513



Gambar 20 Model Confusion Matrix menggunakan Decision Tree

Model Classifier	Execution Time (seconds)	Accuracy
Neural Network with Dropout (ReLU)	128.2	0.997
Decision Tree	133.6	0.117

Tabel 7: Hasil dari pengujian dunia nyata

6. KESIMPULAN DAN SARAN

Dalam penelitian ini, ditunjukkan bahwa penggunaan jaringan saraf dalam untuk deteksi malware statis dapat dilakukan dan berpotensi untuk peningkatan lebih lanjut. Eksperimen dalam penelitian ioni menunjukkan bahwa bahkan dalam situasi yang melibatkan data terstruktur, penggunaan jaringan saraf masih bisa lebih efisien dibandingkan dengan pohon keputusan. Metode vektorisasi file yang dapat meringkas file besar secara efektif untuk klasifikasi ditetapkan dalam penelitian ini. Pentingnya ketersediaan dataset yang besar di domain tersebut tidak dapat diabaikan. Penelitian ini menunjukkan bahwa analisis malware statis dapat menjadi alat yang efektif dalam klasifikasi malware terlepas dari keberadaan dan tingkat deteksi analisis malware dinamis. Untuk saran dimasa depan, diperlukan penelitian lebih lanjut di bidang ini untuk menentukan seberapa efisien jaringan saraf bisa menjadi pengklasifikasi untuk data terstruktur dibandingkan dengan model pohon keputusan. Pengujian dunia nyata menunjukkan bahwa masih ada celah di area ini yang belum dieksplorasi dan akan memerlukan pengujian lebih lanjut untuk implementasi praktis

DAFTAR PUSTAKA

Ammar AE Elhadi, Mohd A Maarof, and Ahmed H Osman. Malware detection based on hybrid signature behaviour application programming interface call graph. American Journal of Applied Sciences, 9(3):283, 2012.

Anaconda. Anaconda software distribution version 2-2.4.0, November 2016.

Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In Twenty-Third Annual Computer Security Applications Con-ference (ACSAC 2007), pages 421-430. IEEE, 2007.

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In Australasian Joint Conference on Artificial Intelligence, pages 137-149. Springer, 2016.

Byron P Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 543(2-3):577-584, 2005.

Charles E Metz. Basic principles of roc analysis. In Seminars in nuclear medicine, volume 8, pages 283-298. Elsevier, 1978.

Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to information retrieval. Natural Language Engineering, 16(1):100-103, 2010.

Daniel Bilar. Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics, 1(2):156-168, 2007.

David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware. In Botnet Detection, pages 65-88. Springer, 2008.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Dilshan Keragala. Detecting malware and sandbox evasion techniques. SANS Institute InfoSec Reading Room, 16, 2016.

Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. Journal of machine learning research, 12(Oct):2825-2830, 2011.

Francois Chollet et al. Keras. https://keras.io, 2015.

Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Control flow graphs as malware signatures. In International workshop on the Theory of Computer Viruses, 2007.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly effcient gradient boosting decision tree. In Advances in Neural Information Processing Systems, pages 3146-3154, 2017.

Hamid Divandari, Bassir Pechaz, and Majid Vafaie Jahan. Malware detection using markov blanket based on opcode sequences. In 2015 International Congress on

Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. Opem: A static-dynamic approach for machine-learning-based malware detection. In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, pages 271-280. Springer, 2013.

J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90-95, 2007.

Jeremy Z Kolter and Marcus A Maloof. Learning to detect malicious executables in the wild. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 470-478. ACM, 2004.

J-Michael Roberts. Virus share.(2011). URL https://virusshare.com, 2011.

John Nickolls, Ian Buck, and Michael Garland. Scalable parallel programming. In 2008 IEEE Hot Chips 20 Symposium (HCS), pages 40-53. IEEE, 2008.

Jon Oberheide, Michael Bailey, and Farnam Jahanian. Polypack: an automated online packing service for optimal antivirus evasion. In Proceedings of the 3rd USENIX conference on Offensive technologies, pages 9. USENIX Association, 2009.

Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pages 11-20. IEEE, 2015.

Karthik Raman et al. Selecting features to classify malware. InfoSec Southwest, 2012.

Katherine Heller, Krysta Svore, Angelos D Keromytis, and Salvatore Stolfo. One class support vector machines for detecting anomalous windows registry accesses. In ICDM Workshop on Data Mining for Computer Security, 2003.

Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. Feature hashing for large scale multitask learning. arXiv preprint arXiv:0902.2206, 2009.

M. Sikorski and A. Honig. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012.

Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR), 44(2):6, 2012.

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensor ow: A system for large-scale machine learning. In 12th fUSENIXg Symposium on Operating Systems Design and Implementation (fOSDIg 16), pages 265-283, 2016.

Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In Proceedings of the 39th annual meeting on association for computational linguistics, pages 26-33. Association for Computational Linguistics 2001.

Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. Technical report, WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES, 2006.

Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. ACM SIGPLAN Notices, 42(1):377- 388, 2007.

Naman Bagga. Measuring the effectiveness of generic malware models. Master's thesis, San Jose State University, 2017.

Philip OKane, Sakir Sezer, and Kieran McLaughlin. Obfuscation: The hidden malware. IEEE Security & Privacy, 9(5):41-47, 2011.

Randy Kath. The portable executable file format from top to bottom. MSDN Library, Microsoft Corporation, 1993.

Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. Malware classification with recurrent networks. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1916-1920. IEEE, 2015.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning, pages 161-168. ACM, 2006.

Robert E Schapire. The boosting approach to machine learning: An overview. In Nonlinear estimation and classification, pages 149-171. Springer, 2003.

Romain Thomas. Lief - library to instrument executable formats. https://lief.quarkslab.com/, April 2017.

Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi.

Srilatha Attaluri, Scott McGhee, and Mark Stamp. Profile hidden markov models and metamorphic virus detection. Journal in computer virology, 5(2):151-169, 2009.

p-ISSN : 2808-8751 e-ISSN : 2798-2513

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for effcient numerical computation. Computing in Science & Engineering, 13(2):22, 2011.

T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagation for classification. International Journal of Computer Theory and Engineering, 3(1):1793-8201, 2011.

Technology, Communication and Knowledge (ICTCK), pages 564-569. IEEE, 2015.

Tom Fawcett. An introduction to roc analysis. Pattern recognition letters, 27(8):861-874, 2006.

Travis E Oliphant. Python for scientic computing. Computing in Science & Engineering, 9(3):10-20, 2007.

Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. Statistics and its Interface, 2(3):349-360, 2009.

Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS '14, pages 247-252, New York, NY, USA, 2014. ACM.

Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 399-418. Springer, 2016.

Wes McKinney et al. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference, volume 445, pages 51-56. Austin,TX, 2010.

Wikimedia Commons. Portable executable 32 bit structure in svg fixed, 2016. https://commons.wikimedia.org/wiki/File: Portable Executable 32 bit Structure in SVG fixed.svg.