ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

DETEKSI MALWARE ANDROID BERDASARKAN SYSTEM CALL MENGGUNAKAN ALGORTIMA SUPPORT VECTOR MACHINE

Sendi Herlambang[*1], Setio Basuki[2], Denar Regata Akbi[3], Zamah Sari[4]
Teknik Informatika, Universitas Muhammadiyah Malang

Kontak Person:
Sendi Herlambang
Universitas Muhammadiyah Malang
e-mail: sendiherlambang.cisco@gmail.com

Abstrak

Android merupakan salah satu sistem operasi berbasis linux pada smartphone yang memiliki banyak fungsi sehingga dapat membantu kinerja penggunanya. Kelebihan Android adalah bersifat open source code sehingga memudahkan para pengembang untuk membuat dan memodifikasi aplikasi yang belum tersedia di sistem operasi Android sesuai dengan kebutuhan. Seiring dengan perkembangan teknologi, juga memicu berkembangnya file-file jahat atau biasa disebut dengan malware. Aktivitas berbahaya yang diakibatkan oleh malware sangat merugikan bagi para korbannya, sehingga dibutuhkan cara untuk mengatasinya. Umumnya pendeteksian malware menggunakan metode signature-basedakan tetapi metode ini mudah untuk dikelabui oleh malware yang memiliki kemampuan polimorfik. Penelitian ini bertujuan untuk melakukan pendeteksian jenis malware Android secara Dinamis dan melakukan klasifikasi jenis malware menggunakan algoritma Support Vector Machine (SVM). Metode pada penelitian ini dibagi menjadi beberapa tahapan yaitu pengumpulan data penelitian, pengambilan informasi dari system call, pra-proses data, seleksi fitur, pelatihan data, pengolahan data dalam pengujian, dan pengujian klasifikasi. Berdasarkan dari penelitian ini menunjukkan hasil akurasi yang cukup tinggi, yaitu dataset tanpa seleksi fitur sebesar75.5556% dan dataset dengan metode seleksi fitur Wrapper SubsetEval Sebesar 73.3333%. Dapat disimpulkan dari akurasi tersebut bahwa system call dapat digunakan sebagai pengembangan klasifikasi jenis malware Android untuk ke depannya.

Kata kunci: *Android, Malware, SVM, System Call*

1. Pendahuluan

Android merupakan salah satu sistem operasi berbasis linux pada *smartphone* yang banyak digunakan karena memiliki berbagai fungsi yang dapat memudahkan pekerjaan. Sistem operasi Android dapat digambarkan sebagai jembatan antara *smartphone* dan penggunanya, sehingga pengguna dapat mengoperasikan *smartphone* dan dapat menggunakan aplikasi yang tersedia. *Smartphone* berbasis Android sangat membantu *user* dalam melakukan berbagai aktivitas seperti berkomunikasi, berbelanja secara *online*, presentasi, dan lain sebagainya. Keunggulan system operasi Android dibanding dengan sistem operasi *smartphone* lainnya adalah Android bersifat *open source code* sehingga memudahkan para pengembang untuk membuat dan memodifikasi aplikasi atau fitur–fitur yang belum terdapat pada sistem operasi Android sesuai dengan kebutuhan. Seiring dengan perkembangan teknologi, tidak bisa dihindari berkembangnya file-file jahat atau biasa disebut dengan *malware*.

Malware (*malicious software*) adalah program komputer atau perangkat lunak yang diciptakan dengan tujuan mencari kelemahan atau bahkan merusak *software* atau sistem operasi[1], *Malware* ada dalam berbagai bentuk seperti *script*, *code*, *active content*, dan perangkat lunak. Secara eksplisit, *malware* merupakan perangkat lunak yang didesain untuk melakukan aktifitas yang dapat merusak aplikasi lainnya, seperti *Trojan*, Virus, *Spyware* dan *Exploit* [2]. *Malware* biasanya menyusup ke dalam suatu program atau aplikasi dan merusak sistem Android bahkan mencuri file-file penting yang ada dalam perangkat *smartphone*. Hal ini menyebabkan banyak pengguna yang merasa terganggu bahkan merasa dirugikan oleh adanya aktivitas berbahaya dari *malware*.

Aktivitas berbahaya yang diakibatkan oleh *malware* sangat merugikan bagi para korbannya, mulai dari kerusakan pada sistem operasi Android, informasi pribadi yang dicuri, hingga aktivitas yang diintai dan disadap, sehingga dibutuhkan cara untuk mengatasinya. *System call* adalah suatu cara bagi aplikasi untuk meminta *service* dari kernel sistem operasi, setiap aplikasi pasti memanggil *system call*. Oleh karena itu, dengan mengamati *system call* tingkah laku dari suatu aplikasi ataupun *malware*

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

dapat dilihat [3]. *Malware* yang sedang beroperasi pada system Android pun bisa dilihat tingkah lakunya dari *system call* yang didapat.

Ada dua cara untuk melakukan pendeteksian *malware*, yaitu secara statis dan dinamis, Salah satu pendeteksian secara dinamis dapat dilakukan dengan mengamati *system call*. Setelah system call dari aplikasi sudah didapatkan maka akan dilakukan klasifikasi untuk menentukan jenis malware tersebut. Pada saat ini telah terdapat beberapa penelitian tentang klasifikasi *malware* pada android seperti pada penelitian [4][5][6][7][8]. Pada penelitian [4] peneliti menggunakan *permesion* untuk melihat perilaku perangkat lunak, pada penelitian [5] menggunakan *Android manifest* untuk mengambil informasi perangkat lunak, pada penelitian [6] menggunakan *Subgraph* untuk klasifikasi, pada penelitian [7] menggunakan *system call* untuk melihat perilaku dari perangkat lunak Android, pada penelitian [8] menggunakan *system call graph* untuk melihat perilaku dari perangkat lunak. Uraian tersebut menunjukkan belum adanya penelitian yang menggunakan *system call* untuk mendapat informasi dari perilaku perangkat lunak serta melakukan klasifikasi jenis *malware* dengan menggunakan SVM (*Support Vector Machine*). SVM merupakan suatu Teknik klasifikasi, yang dimana cara kerjanya yaitu menemukan *hyperplane* yang bisa memisahkan dua set data atau lebih dari dua kelas atau lebih yang berbeda [3]. SVM memiliki keunggulan yaitu adalah dalam menentukan jarak menggunakan *support vector* sehingga proses komputasi menjadi cepat [9]. Menurut [10], SVM merupakan metode klasifikasi yang memiliki proses pelatihan yang efisien dan dapat dioptimisasi pada segala bidang percobaan. Sehingga pada penelitian ini, akan diimplementasikan *system call* untuk mendapatkan informasi dari perangkat lunak dan menggunakan algoritma *Support Vector Machine* untuk melakukan klasifikasi.

Penelitian ini bertujuan untuk melakukan klasifikasi jenis *malware* menggunakan algoritma *Support Vector Machine* (SVM). Data *malware* yang didapat berupa file .apk dari situs *virusshare.com.* Perintah yang digunakan untuk menangkap aktifitas yang terjadi selama aplikasi berjalan pada Android menggunakan perintah *strace*, kemudian disusun dalam sebuah matriks. Pengujian klasifikasi SVM antara data latih dan data uji sehingga menghasilkan *output* berupa analisis. Perhitungan akurasi dilakukan dengan cara membagi antara jumlah data uji benar dengan jumlah data uji, kemudian dikali 100% maka akan didapatkan hasil akurasinya.

2. Metode Penelitian

Metode pada penelitian ini dibagi menjadi beberapa tahapan yaitu pengumpulan data penelitian, pengambilan informasi dari *system call*, praproses data, seleksi fitur, pelatihan data, pengolahan data dalam pengujian, dan pengujian klasifikasi. Berikut adalah uraian masing-masing tahapan.

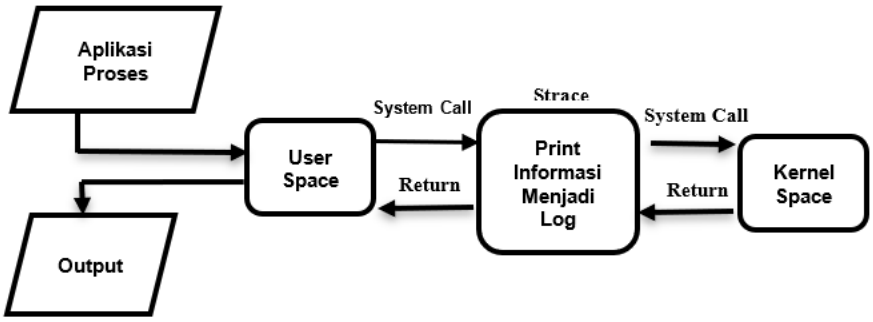
Tahap pertama adalah pengumpulan data. Pengumpulan data merupakan proses yang bertanggung jawab dalam pengunduhan aplikasi *malware* dari situs *www.virusshare.com*, kemudian untuk mengetahui jenis *malware* menggunakan situs www.virustotal.com. Berdasarkan data yang diperoleh, didapatkan banyak aplikasi yang mengandung *malware* jenis *Adware*, *Droidkunfu*, *Plankton*, dan *Trojan*. Sehingga pada penelitian kali ini akan digunakan 4 jenis *malware* tersebut sebagai kelas dalam permodelan klasifikasi. Parameter yang digunakan adalah *system call* dari aplikasi *malware* yang telah di instal pada *platform* Android.

Tahap kedua adalah pengambilan informasi dari*System Call*.*System Call* melibatkan aplikasi yang menggunakan fungsi *library glibc*. Data yang sudah dikumpulkan pada tahap pertama kemudian dieksekusi untuk mengambil informasi data dari *system call* aplikasi dengan menggunakan perintah *strace* yang berfungsi untuk menangkap frekuensi dan PID proses yang sedang berjalan. Aplikasi dijalankan selama 15 detik, ketika aplikasi dijalankan informasi *system call* akan terekam dan tersimpan pada file .txt. Dari informasi tersebut didapatkan informasi berupa *system call* yang dipanggil dan frekuensinya. Kemudian, informasi tersebut diubah kedalam bentuk matriks untuk proses identifikasi menggunakan algoritma *Support Vector Machine* pada tahap selanjutnya. Gambar 1 adalah skema proses pengambilan informasi *system call*.

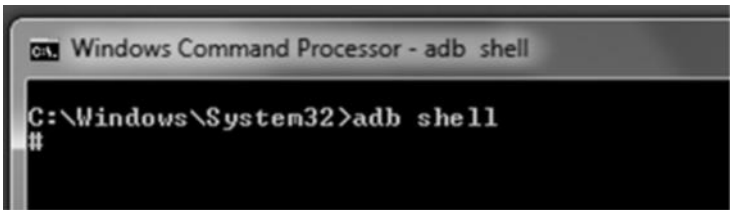
Semua aplikasi di instal pada *smartphone* Android yang sudah di-*root* dan ditambahkan program *strace* untuk merekam pemanggilan *systemcall*. Kemudian *smartphone* dihubungkan dengan komputer yang sudah terpasang program *Android Debug Bridge* (ADB) menggunakan kabel USB. *Systemcall* yang digunakan oleh aplikasi *malware* Android pada saat dijalankan dapat direkam pada

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

komputer. Setiap aplikasi pada penelitian ini dijalankan selama 15 detik. Pada komputer program adb dijalankan dengan menggunakan perintah *shell* seperti yang ditunjukkan pada Gambar 2.



Gambar 1 Proses pengambilan informasi *system call*

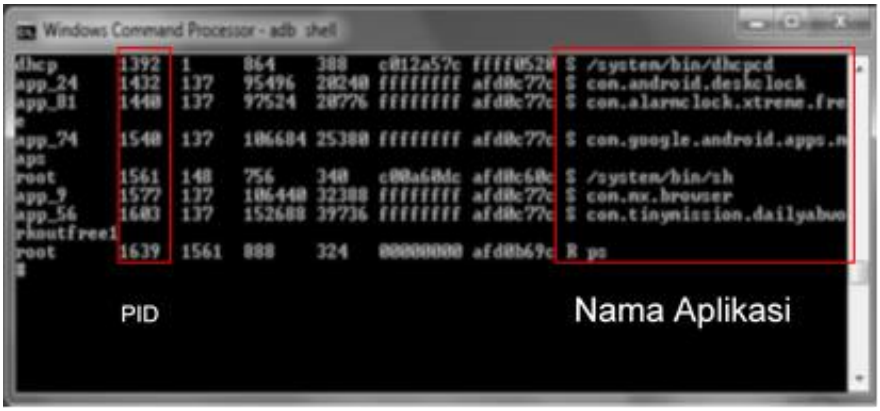


Gambar 2 Perintah untuk memulai program ADB.

Program ADB yang berhasil dijalankan menunjukkan *shell linux* pada perangkat Android dapat dikendalikan melalui komputer. Langkah selanjutnya adalah menampilkan *Process ID*(PID) dari aplikasi yang akan diamati, untuk menampilkannya menggunakan perintah *ps* seperti pada Gambar 3.

Gambar 3 Perintah untuk menampilkan PID

Langkah selanjutnya adalah mencari PID dari aplikasi yang akan diamati. Gambar 5 adalah gambar PID dari aplikasi yang akan diamati. Keterangan nama aplikasi berada pada kolom ke-9, sedangkan PID berada pada kolom ke-2. Untuk merekam system call adalah dengan memanggil PID dari aplikasi yang berjalan, sesuaikan nama aplikasi dan PID dari aplikasi tersebut.



Gambar 5 PID dari aplikasi yang akan di amati

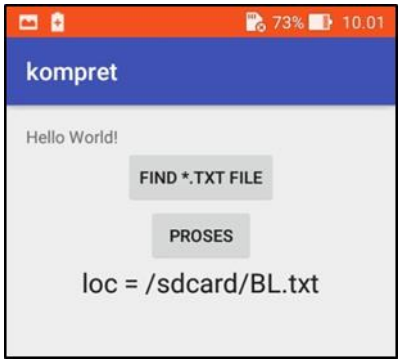
ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

Selanjutnya untuk merekam *system call* yang akan dipanggil dari salah satu aplikasi, membutuhkan jendela baru dari dari program ADB shell. Pada jendela baru tersebut menggunakan perintah *strace -p PID &> /sdcard/nama.txt* untuk merekam pemanggilan *system call* dan disimpan ke dalam file berformat .txt. Pada Gambar 6 adalah contoh dari system call yang telah dipanggil dan disimpan ke dalam file .txt.

```
getuid32()                              = 10116
SYS_329(0x11, 0xbec2ce98, 0x10, 0, 0)   = 1
read(14, "\1\0\0\0\0\0\0\0", 8)         = 8
clock_gettime(CLOCK_MONOTONIC, {80186, 491794752}) = 0
SYS_329(0x11, 0xbec2ce98, 0x10, 0, 0)   = 0
clock_gettime(CLOCK_MONOTONIC, {80186, 492173658}) = 0
SYS_329(0x11, 0xbec2ce98, 0x10, 0xffffffff, 0) = 1
read(14, "\1\0\0\0\0\0\0\0", 8)         = 8
clock_gettime(CLOCK_MONOTONIC, {80196, 460484936}) = 0
getuid32()                              = 10116
SYS_329(0x11, 0xbec2ce98, 0x10, 0, 0)   = 0
clock_gettime(CLOCK_MONOTONIC, {80196, 462708395}) = 0
SYS_329(0x11, 0xbec2ce98, 0x10, 0, 0)   = 0
clock_gettime(CLOCK_MONOTONIC, {80196, 463965253}) = 0
SYS_329(0x11, 0xbec2ce98, 0x10, 0xffffffff, 0 <unfinished ...>
Process 30323 detached
```

Gambar 6 *System call* yang dipanggil dan disimpan ke dalam file .txt

Tahap selanjutnya adalah pra-proses data. Setelah tersimpan file berisi *system call* tersebut, selanjutnya diubah menjadi matrik berformat .csv menggunakan program sederhana yang telah dibuat. Program tersebut dibuat menggunakan Android Studio dan berjalan pada *platform* Android karena hasil dari file .txt yang berisi informasi tentang *system call* tersimpan di dalam perangkat Android. Gambar 7 adalah tampilan dari program sederhana untuk mengubah file .txt menjadi file .csv.



Gambar 7 Tampilan program untuk mengubah file .txt menjadi file .csv

Menginputkan file *system call* dalam bentuk .txt ke dalam program dengan memilih tombol FIND*.TXT FILE, selanjutnya klik pada tombol "proses" lalu program akan mengubah file .txt menjadi file .csv dalam bentuk matrik. Hasil keluaran dari program tersebut seperti pada Tabel 1.

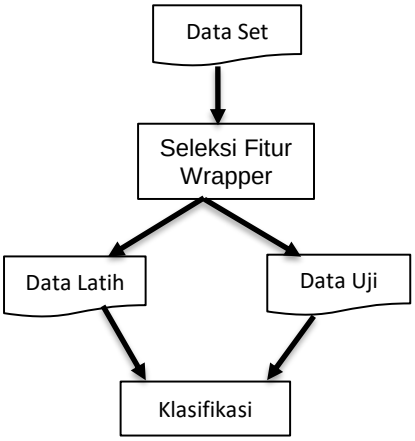
Tabel 1 File *system call* berupa matrik

Epoll_pwait	Futex	Getuid32	Clock_gettime	Jenis
529	93	1025	169	*Adware*
600	670	251	14	*Adware*
772	840	1051	166	*Trojan*
435	2277	61	75	*Trojan*
913	140	1213	2830	*Plankton*
882	894	2861	1467	*Plankton*
14946	333	2702	366	*DroidKungfu*
12879	343	2113	2902	*DroidKungfu*

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

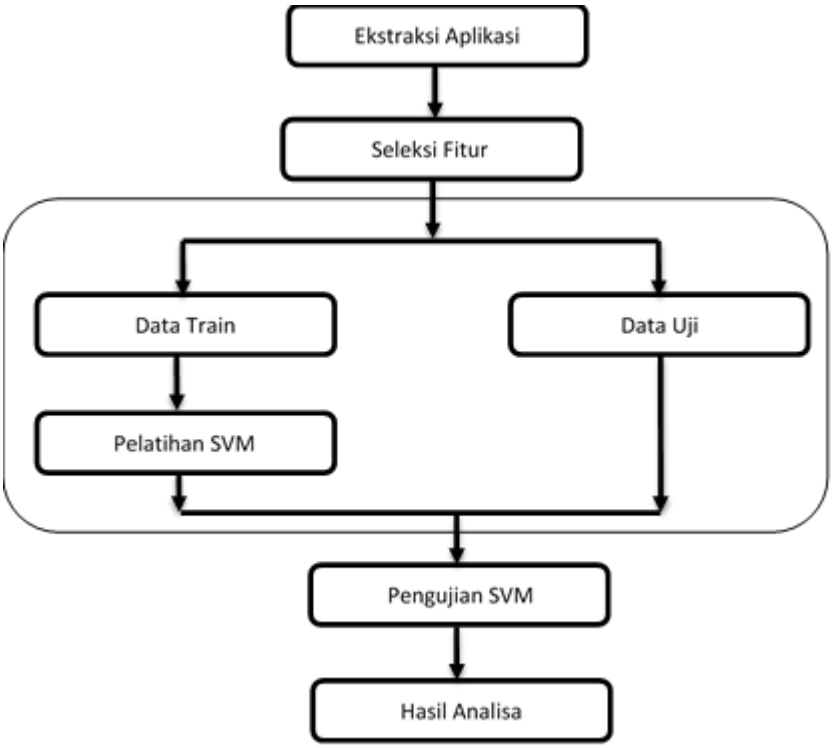
Setelah seluruh data *system call* menjadi file .csv berbentuk matrik, selanjutnya mengolah 70% data atau sebanyak 420 data tersebut menjadi data latih(*data training*) dan 30% data atau sebanyak 180 data menjadi data uji(*data test*).Dari hasil matrik *system call* tersebut dapat diambil fitur sebanyak 48 fitur untuk keperluan klasifikasi. Satu baris mewakili satu fitur dan frekuensinya sedangkan satu kolom mewakili satu aplikasi Android dan baris terakhir adalah jenis dari *malware* Android. Angka pada *system call* adalah nilai frekuensi selama *system call* dipanggil. Pengolahan data tersebut menggunakan aplikasi *weka* dan menggunakan algoritma *Support Vector Machine* untuk data latih.

Tahap selanjutnya adalah seleksi fitur. Teknik seleksi fitur yang akan digunakan pada penelitian ini adalah *Wrapper Subset Evaluation* dengan metode *Best First*. Gambar 8 adalah alur proses seleksi fitur.



Gambar 8 Alur seleksi fitur

Tahapan selanjutnya adalah pelatihan data. Pada penelitian ini data akan dilatih dengan algoritma *Support Vector Machine* menggunakan aplikasi *weka*. Seluruh data *system call* yang sudah terkumpul diubah menjadi format .csv atau .arff untuk keperluan *training* di aplikasi *Weka*. Gambar 9 adalah arsitektur sistem untuk klasifikasi.



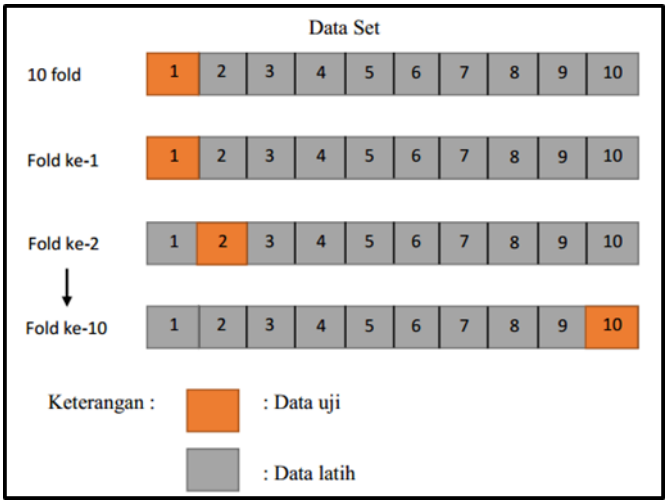
Gambar 9 Arsitektur sistem untuk klasifikasi

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

Berikut adalah uraian proses pada tahap ini:

1. Data latih diperoleh dari hasil pengamatan system call berupa matriks aplikasi malware yang telah di panggil system call nya. Dari keseluruhan data, 70% digunakan sebagai data latih.
2. Data Uji diperoleh dari hasil pengamatan system call berupa matriks aplikasi malware. Dari keseluruhan data, 30% digunakan sebagai data Uji.
3. Pelatihan SVM adalah sebuah proses pelatihan data latih, dimana data system call berupa matriks yang dihasilkan akan diformulasikan dalam rumus SVM.
4. Pengujian SVM sebuah proses pengujian klasifikasi antara data latih dengan data uji, dengan cara menghitung keakuratan dari akurasi klasifikasi masing-masing kelas, sehingga menghasilkan output berupa analisis.
5. Pengujian SVM adalah sebuah proses pengujian klasifikasi antara data latih dengan data uji sehingga menghasilkan *output* berupa analisis.

Tahap selanjutnya adalah pengolahan data dalam pengujian. Pengujian data dalam penelitian ini menggunakan metode pengujian *K-Fold Cross Validation*. Seluruh data malware yang ada akan dibagi menjadi sepuluh subset, yaitu: *fold 1, fold 2, fold 3, fold 4, fold 5, fold 6, fold 7, fold 8, fold 9, dan fold 10*. Hal tersebut dilakukan dengan tujuan untuk mencari performansi terbaik. Pelatihan akan dilakukan secara berulang sebanyak sepuluh kali, setiap pengulangan sembilan *fold* akan dijadikan data latih dan satu *fold* akan dijadikan data uji. Proses ini dilakukan sampai semua *fold* pernah berperan sebagai data latih dan data uji. Proses pembagian data latih dan data uji dalam proses 10-*fold cross validation* terdapat pada Gambar 10.



Gambar 10 Contoh iterasi data dengan *k-fold cross validation*

Cara kerja *K-fold cross validation* adalah sebagai berikut:

1. Total *instance* dibagi menjadi N bagian.
2. *Fold* ke-1 adalah ketika bagian ke-1 menjadi data uji (*testing* data*)* dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut. Perhitungan akurasi tersebut dengan menggunakan Persamaan 1.

$$akurasi = \frac{jumlahdataujibenar}{jumlahdatauji} x100 \quad (1)$$

3. *Fold* ke-2 adalah ketika bagian ke-2 menjadi data uji (*testing data*) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut.
4. Demikian seterusnya hingga mencapai *fold* ke-K. Hitung rata-rata akurasi dari K buah akurasi di atas. Rata-rata akurasi ini menjadi akurasi final.

Tahap terakhir adalah pengujian klasifikasi. Pengujian klasifikasi bertujuan untuk mengetahui keberhasilan system klasifikasi. Rumus yang digunakan adalah Persamaan 2.

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

$$akurasi = \frac{jumlahdataujibenar}{jumlahdatauji} x100 \quad (2)$$

Berikut adalah Persamaan 3, yang merupakan perhitungan akurasi dari data uji yang telah diprediksi oleh algoritma SVM.

$$akurasi = \frac{136}{180} x100 = 75.5556 \quad (3)$$

Hasil akurasi prediksi yang didapat sebesar 75.5556%.

3. Hasil Analisa Data

Hasil analisa data pada penelitian ini dibagi menjadi empat bahasan, yaitu hasil persiapan data, hasil pre-processing data, hasil *cross validation* dan hasil pengujian klasifikasi dengan menggunakan algoritma *Support Vector Machine*.

3.1 Hasil Persiapan Data

Pada bagian ini menggunakan 4 jenis *Malware* Android diantaranya adalah *Adware, Trojan, Plankton*, dan *DroidKungfu* dengan format .apk, total keseluruhan data sebanyak 600 data. Dari keseluruhan data, 70% digunakan sebagai data latih dan 30% digunakan sebagai data uji dama model klasifikasi (Tabel 2).

Tabel 2 Pembagian Jumlah Data *Malware*

No.	Jenis *Malware*	Jumlah	Data Latih 70%	Data Uji 30%
1	*Adware*	99	75	24
2	*Trojan*	263	176	87
3	*Plankton*	88	64	24
4	*DroidKungfu*	150	105	45
	Total	600	420	180

3.2 Hasil Pre-processing Data

Setelah file .txt diproses menggunakan program, hasil keluaran dari program tersebut menjadi matrks berformat .csv seperti pada Tabel 3 yang nantinya akan digunakan untuk keperluan klasifikasi pada Weka.

Tabel 3 Hasil keluaran dari program menjadi matriks

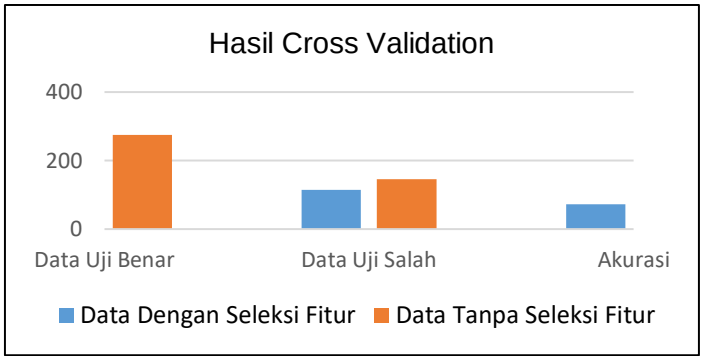
epoll_pwait	futex	getuid32	clock_gettime	Jenis
529	93	1025	169	*Adware*
600	670	251	14	*Adware*
772	840	1051	166	*Trojan*
435	2277	61	75	*Trojan*
913	140	1213	2830	*Plankton*
882	894	2861	1467	*Plankton*
14946	333	2702	366	*DroidKungfu*
11287	343	2113	2902	*DroidKungfu*

Angka pada baris adalah frekuensi atau banyaknya *system call* yang dipanggil selama aplikasi berjalan sedangkan *epoll_pwait, futex, getuid32, clock_gettime* adalah nama dari *system call* yang ada pada aplikasi ketika berjalan dan direkam oleh program *adb shell*. Fitur yang didapat dari seluruh aplikasi berjumlah 48 fitur, dan tidak semua fitur berguna dalam proses klasifikasi. Oleh karena itu digunakan teknik seleksi fitur untuk mengurangi fitur yang tidak diperlukan dalam proses klasifikasi menggunakan algoritma SVM.

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

3.3 Hasil *Cross Validation*

Pada penelitian ini jumlah *fold* yang digunakan untuk keperluan *cross validation* adalah sebanyak 10 *fold*. Sehingga terdapat 10 kali percobaan untuk mendapatkan akurasi dari data *training*. Dari hasil akurasi yang didapatkan adalah sebesar 56.90476% dengan menggunakan data yang sudah dilakukan seleksi fitur sedangkan data yang tidak dilakukan seleksi fitur adalah sebesar 57.38095%. Gambar 11 merupakan hasil akurasi data *training* yang lebih besar adalah data *training* yang sudah dilakukan seleksi fitur, sehingga untuk keperluan klasifikasi nantinya akan menggunakan data training yang sudah dilakukan seleksi fitur.

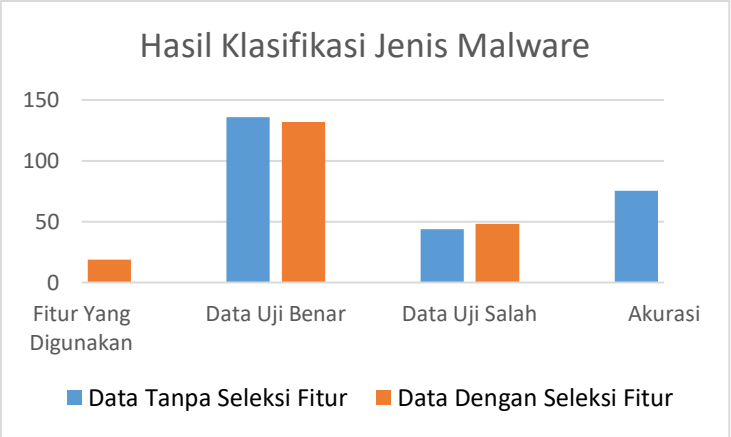


Gambar 11 Grafik Hasil *Cross Validation*

3.4 Hasil Analisa dari Proses Klasifikasi

Berdasarkan hasil pengujian klasifikasi menggunakan algoritma *Support Vector Machine* dengan *Kernel Polynomial* didapatkan Gambar 12 dan hasil sebagai berikut.

1. Dataset tanpa seleksi fitur
   Hasil klasifikasi menggunakan algoritma SVM dengan menggunakan semua fitur yang didapat dari pengumpulan data yaitu sebanyak 48 fitur dengan jumlah data uji benar sebanyak 136 data menghasilkan akurasi sebesar 75.5556%.
2. Dataset dengan seleksi fitur *Wrapper SubsetEval*
   Hasil klasifikasi menggunakan algoritma SVM dengan menggunakan metode seleksi fitur *Wrapper SubsetEval* menyisakan 19 fitur dengan jumlah data uji benar sebanyak 132 data menghasilkan akurasi sebesar 73.3333%.



Gambar 12 Grafik Hasil Klasifikasi Jenis *Malware*

4.Kesimpulan

Berdasarkan dari penelitian ini menunjukkan hasil akurasi yang cukup tinggi, yaitu dataset tanpa seleksi fitur sebesar75.5556% dan dataset dengan metode seleksi fitur *Wrapper SubsetEval* Sebesar73.3333%. Dapat disimpulkan dari akurasi tersebut bahwa *system call* dapat digunakan sebagai pengembangan klasifikasi jenis malware Android untuk ke depannya. Hanya saja akurasi yang

ISSN (Cetak) 2527-6042
eISSN (Online) 2527-6050

didapat belum mencapai hasil yang maksimal. Hasil akurasi yang tidak jauh berbeda antara dataset yang menggunakan metode seleksi fitur dengan dataset yang tidak menggunakan metode seleksi fitur. Seleksi fitur dapat mengurangi jumlah fitur yang akan digunakan untuk keperluan klasifikasi, tetapi tidak meningkatkan hasil akurasi secara signifikan.

Referensi

[1] Yunus, Mohammad. 2009. Pendeteksian Malware Dengan Menggunakan Algoritma Multi-Naive Bayes. *Tugas Akhir*. Jurusan Teknik Informatika Institut Teknologi Sepuluh November.
[2] Kramer, S., & Bradfield, J. C. 2009. A General Definition of Malware. *Journal in Computer Virology*. Vol. 6, no. 2. pp 105–114.
[3] Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X., Wang, X., Antipolis, S. (n.d.). 2009. *Effective and Efficient Malware Detection at the End Host*. USENIX Association Berkeley. Montreal, Canada.
[4] Baltaci Nuray, dkk. 2014. *The Analysis of Feature Selection Methods and Classification Algorithms in Permission Based Android Malware Detection*. Cyber Defense and Security Laboratory of METU-COMODO, Informatics Institute Middle East Technical University (METU), Ankara, Turkey.
[5] Li Xiang, dkk. 2016. *AN Android Malware Detection Method Based on Androidmanifest File*. Beijing University of Posts and Telecommunications, Beijing 100876, China.
[6] Fan Ming, dkk. 2016. *Frequent Subgraph based Familial Classification of Android Malware.* Department of Computer, The Hong Kong Polytechnic University, 999077, China.
[7] Xiao Xi, dkk. *Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences*. 1Graduate School at Shenzhen, Tsinghua University, 518055 Shenzhen, People's Republic of China.
[8] Shifu Hou, dkk. 2016. *Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs*. Department of Computer Science and Electrical Engineering West Virginia University Morgantown, WV, 26506, USA.
[9] Vapnik, V dan Cortes, C. 1995. *Support Vector Networks*. Machine Learning, 20, 273-297.
[10] Cristianini N, Taylor JS. (2000). *An Introduction to Support Vector Machine and Other Kernel-based Learning Methods*. Cambridge (GB): Cambridge University Press.