Deteksi Malware dengan Menggunakan API Calls

William Halim - 18217021

Sistem dan Teknologi Informasi - Institut Teknologi Bandung

Abstract – Sebagai salah satu *mobile operating system* paling populer di kalangan masyarakat, Android menarik perhatian para *attacker* untuk menginjeksi *malware* pada serangkaian aplikasi – aplikasi berbasis *mobile*. Untuk menangani serangkaian serangan *mobile malware*, dibutuhkan pemahaman terhadap bagaimana *permissions* bekerja dan dinyatakan pada aplikasi dan API Calls pada Android. Masih banyak *malware analyst* yang menggunakan cara manual, yaitu dengan melakukan inspeksi terhadap sampel *malware* yang dicurigai dengan mengambil bagian tertentu dari suatu *software* dan menganalisis bukti keberadaan *malware* pada kode – kode pada *software* tersebut. Pada makalah ini, akan dijelaskan metode dan pendekatan efektif untuk melakukan deteksi terhadap *malware*, yaitu dengan mengekstrasi API *call sequence* dari program *malware* yang telah ditemukan sebelumnya

Index term – Android, Malware, Permissions, Api Call

I. Pendahuluan

I.1. Latar Belakang

Malware (*Malicious Software*) adalah perangkat lunak yang didesain dan dikembangkan untuk memberikan kerusakan terhadap perangkat – perangkat elektronik, seperti komputer dan *smartphone*. Serangan berupa *malware* ini seringkali bertujuan untuk memperoleh informasi personal secara ilegal. Adapun serangan – serangan tersebut dilancarkan ketika *user* sedang melakukan *browsing* di internet ataupun men-*download* suatu aplikasi yang ternyata di dalamnya

terdapat *malware* yang tidak terdeteksi oleh *detection tool* pada perangkat yang dimiliki oleh *user*.

Akhir – akhir ini, seiring dengan tingginya popularitas *smartphone* di kalangan masyarakat, beberapa *attacker* menjadikan perangkat *smartphone* ini sebagai target untuk serangkaian serangan berupa injeksi *malware* melalui aplikasi – aplikasi berbasis *mobile* yang diunduh oleh pengguna. Mayoritas *smartphone* yang dijual di pasaran menggunakan Android *Operating System*. Berdasarkan penelitian, Android Operating System ditemukan di hampir lebih dari setengah smartphone yang diproduksi di seluruh dunia. Statistik menunjukkan bahwa 85,2 persen dari 1,276 miliar *smartphone* yang diproduksi pada tahun 2017 menggunakan Android Operating System [1].

Tingginya penggunaan akses internet melalui aplikasi *mobile* menarik perhatian para *attacker* untuk mencari celah pada sistem. Adapun penyerangan dapat dilakukan dengan dua cara yang umum digunakan, yaitu melalui teknik *fraudulent mobile apps* dan *injected malicious* apps. *Fraudulent* mengacu pada teknik dimana *attacker* menciptakan aplikasi serupa dengan aplikasi yang sudah ada untuk mengelabui user. Melalui aplikasi tersebut, *user* akan memasukkan informasi tertentu yang akan dimanfaatkan oleh *attacker*. Sedangkan, *injected malicious apps* adalah aplikasi yang didalamnya terdapat kode – kode tertentu yang dinjeksi oleh *attacker* yang bertujuan untuk mengacaukan sistem komputer. Hal ini dapat terjadi ketika *user* mengunduh berbagai aplikasi gratis yang kredibilitasnya dapat dipertanyakan.

Pada makalah ini, akan dibahas cara – cara dan metode yang dapat digunakan untuk mendeteksi keberadaan *malware* pada suatu aplikasi yang dinilai mencurigakan.

I.2. Rumusan Masalah

Rumusan Masalah dari makalah ini adalah sebagai berikut :

1. Bagaimana Permissions dan API Call bekerja
2. Bagaimana Malware dapat terdeteksi melalui analisis terhadap pattern dari API Call

3. Bagaimana efektivitas dari Analisis API Call Sequence dalam mendeteksi keberadaan *malware*

I.3. Tujuan

Berikut adalah tujuan dari makalah ini :

1. Menjelaskan bagaimana Permissions dan API Calls bekerja pada Smartphone berbasis Android
2. Menjelaskan bagimana deteksi *malware* dilakukan melalui analisis terhadap *behavior* dan *pattern* dari API Call
3. Menjelaskan efektivias dari analsisis API Call Sequence dalam mendeteksi keberadaan *malware*.

I.4. Metodologi

Tabel 1 Metodologi

No.	Bagian	Deskripsi	Isi Bagian
1.	Pendahuluan	Bagian Pendahuluan menjelaskan apa itu *malware* dan mengapa banyak *attacker* menjadikan *smartphone* dengan *Android Operating System* sebagai target untuk menginjeksi *malware* melalui aplikasi – aplikasi yang dapat diunduh.	- Latar Belakang - Rumusan Masalah - Tujuan - Metodologi
2.	Studi Pustaka	Bagian Studi Pustaka menjelaskan bagaimana	- Permissions pada Android

		Permissions yang berperan sebagai *Access Control* bekerja pada Android, sistematika API Call pada Android, dan Tipe Malware beserta cara kerjanya	- API Calls pada Android - Tipe Malware dan cara kerjanya
3.	Pembahasan	Bagian Pembahasan menjelaskan Latar Belakang Platform yang digunakan untuk melakukan deteksi *malware*, proses deteksi *malware* yang dilakukan, serta hasil evaluasi dari metode deteksi tersebut	- Arsitektur Platform - Analisis *Behavior* dan *Pattern* - Hasil Evaluasi
4.	Penutup	Bagian Penutup berisi kesimpulan dan saran	- Kesimpulan - Saran

II. Studi Pustaka

II.1. Permissions pada Android

Setiap komponen pada Android Operating System mengimplementasikan protokol *access control* dengan mengimplementasikan *permissions*. *Permissions* meminta persetujuan dari *user* untuk memberikan akses kepada suatu aplikasi untuk melakukan sutau fungsi tertentu. Daftar – daftar *Permissions* ditampilkan pada saat instalasi dari suatu aplikasi atau pada saat aplikasi sedang berjalan (*running*). Keputusan untuk memberikan akses (*allow*) atau menolak (*deny*) sepenuhnya berada di tangan *user* yang bersangkutan. Tujuan dari

diberlakukannya sistem Permissions ini adalah untuk melindungi privasi data *user*. Terdapat 4 *level* dari Permissions [2] , yaitu sebagai berikut :

a. *Normal Permission*, yang merupakan Permissions dengan risiko yang rendah, seperti pengaksesan kalkulator dan *clock*.
b. *Dangerous Permission*, yang dianggap memiliki risiko yang cukup tinggi, seperti pengaksesan kamera, SMS, dan *location* melalui GPS
c. *Signature Permission*, yang dianggap memiliki risiko yang paling tinggi. Pada mekanisme *Signature Permission* ini, sistem memberikan *permissions* hanya jika aplikasi yang me-*request permission* tersebut ditandatangani dengan *certificate* yang sama dengan *certificate* yang dimiliki oleh aplikasi yang memberikan *permission*.

Tabel di bawah ini merupakan contoh dari beberapa Android *Permissions* besera level dari *permission*-nya :

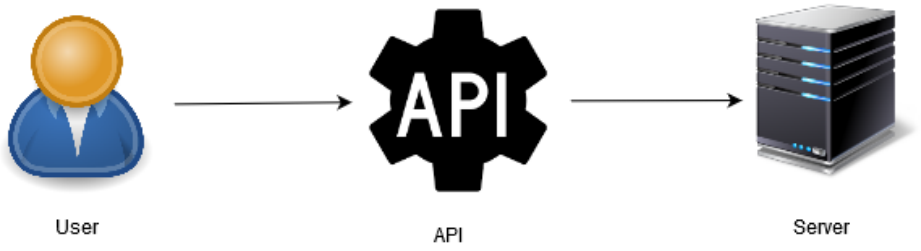
Kategori	*Permission*	Level
Screenlock	• Set_Screenlock • Set_ScreenTime	Normal
Clock	• Set_Alarm • Set_Clock • Set_Timer • Set_Stopwatch	Normal
Bluetooth	• Bluetooth_Admin	Normal
SMS	• Read_SMS • Write_SMS • Receive_Wap_Push	*Dangerous*
Phone	• Call • Answer_Call • Read_Call_Log • Write_Call_Log • Outgoing_Calls	*Dangerous*
Location (GPS)	• Access_Coarse_Location	*Dangerous*

	• Access_Fine_Location	
Contacts	• Read_Contacts • Write_Contacts	*Dangerous*
Storage	• Read_External_Storage • Write_External_Storage	*Dangeorus*

Dengan adanya protokol *access control* tersebut, *user* pengguna Android belum sepenuhnya aman karena *malware* tetap dapat melakukan aktivitas – aktivitas berbahaya tanpa menyatakan *permissions* terhadap *user* yang bersangkutan. Adapun aktivitas berbahaya yang dimaksud dapat berupa meng-*compromise runtime environment* dari Android hingga menyebabkan *crash*.

II.2. API Calls pada Android

API (*Application Programming Interface*) adalah sekumpulan kode yang memungkinkan terjadinya interaksi antara komponen – komponen dalam suatu sistem. Secara sederhana, API berperan sebagai *middleman* antara End User dengan Server dalam suatu sistem. Berikut ini adalah ilustrasi dari cara kerja API secara sederhana :



Gambar 1 Proses API Calls.
Sumber : Dok.Penulis

Ketika End-User mengirimkan *request*, API akan menerima *request* tersebut dan menjalankan instruksi tertentu untuk meneruskan *request* dari *end-user* ke server. API kemudian akan menerima data dari server dan memberikan *response* kepada End-User.

Dalam kaitannya dengan *Malware Detection* pada Android, API Calls memegang peranan yang cukup penting karena pemahaman dan analisis

terhadap kode dan informasi yang terkandung dari API Calls dapat digunakan untuk mencari aktivitas – aktivitas yang berpotensi membahayakan sistem Android. Akan dilakukan investigasi lebih dalam mengenai sistematika *permissions* yang diberikan ketika suatu *malware apps* mengirimkan *request* untuk melakukan API Calls. Hal ini mengingat beberapa API Calls membutuhkan beberapa *permissions* yang masuk ke dalam kategori *dangerous permissions*.

II.3. Obfuscation Techniques dan Tipe Malware

Untuk menghindari *detection tool*, pencipta *malware* menggunakan teknik yang bernama *Obfuscation*. Pada metode *Obfuscation* ini, akan dilakukan *rendering* terhadap *source code* kompleks yang tidak mungkin dapat dipahami Teknik ini digunakan untuk menembus (*bypass*) *code analyzers* yang diciptakan oleh *malware analyst* yang umumnya dapat dibedakan atas Packing, *Polymorphism*, dan *Metamorphism*. *Metamorphic* dan *Polymorphic Malware* mentransformasi kode – kode *malicious* menjadi suatu kode baru tanpa mengubah fungsionalitas dan tujuan awal sehingga alat pendeteksi *malware* tersebut gagal dalam mendeteksi *siganture* dari program yang dibuat oleh *attacker* tersebut [3]. Cascade Virus adalah virus DOS pertama yang menggunakan teknik enkripsi, dimana virus tersebut memulai *pattern*-nya dengan *decryption* yang konstan dan diikuti oleh bagian *body* yang *encrypted*. [4]

Packers biasanya digunakan dalam proses kompresi. *Packers* dapat digunakan untuk melakukan enkripsi file PE (*Portable Executable*) pada *secondary memory* dan dapat me-*restore original executable image* ketika di-*load* ke *main memory* (RAM). Pencipta *Malware* tidak perlu mengubah serangkaian *line of code* untuk mengubah *signature* dari *malware* karena perubahan terhadap *byte sequence* di dalam file PE akan menghasilkan *byte sequence* yang baru dalam PE yang baru diproduksi [5]

Polymorphic Malware menggunakan enkripsi untuk mengubah bagian *body* dari *malware*. Selain itu, *malware* ini juga mengubah *decryption routines* dari

infeksi satu ke infeksi lain selama *encription keys* nya berubah. Hal ini membuat beberapa peneliti *malware* mengembangkan beberapa teknik deteksi yang baru dan berbeda untuk memecahkan *behavior* dan *pattern* dari *malware – malware* yang ada dan bermunculan secara terus menerus. Pada tahun 2006, Symantec Internet Security Threat melaporkan bahwa deteksi terhadap *Polymorphic Malware*, seperti w32.Polip dan w32.Detnat lebih sulit dilakukan dibandingkan dengan tipe *malware* lainnya [6].

Metamorphic Malware mengubah kode tanpa menggunakan enkripsi. Secara umum, terdapat 4 teknik yang digunakan untuk melakukan *obfuscation* secara *metamorphic* [7]. Keempat teknik tersebut adalah sebagai berikut :

a. *Dead-code Insertion*, yang melakukan protokol NOP (*No Operation Performed*)
b. *Code Transposition*, yang mengubah instruksi dengan menggunakan JMPs *instructions* sehingga urutan *instructions* berbeda dari urutan aslinya.
c. *Register Reassignment*, yang menggunakan metode *switching*, dengan mengubah push ebx menjadi push eax to mengubah nama register.
d. *Instruction Substitution*, yang mengganti *instructions* yang ada menjadi *instructions* baru dengan hasil yang sama. Beberapa pembuat *malware* menciptakan *database dictionary* yang berisi kumpulan *instructions* yang ekivalen untuk mempermudah dan mempercepat operasi.

III. Pembahasan

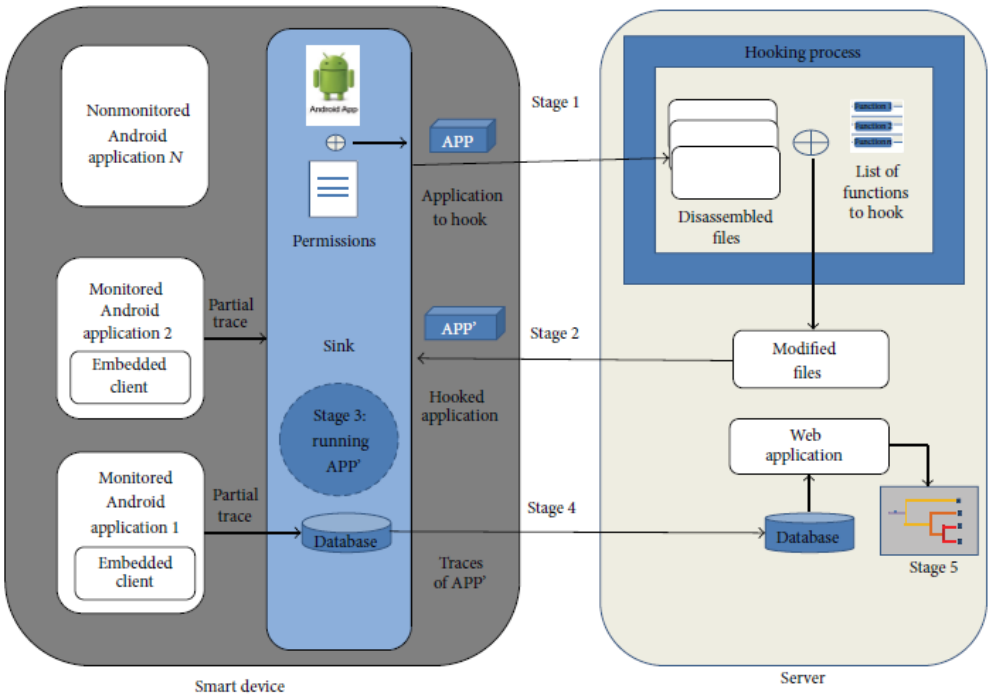
III.1. Latar Belakang Platform

Ketika aplikasi pada Android dijalankan, aplikasi akan memanggil kumpulan *functions* yang didefinisikan oleh *developer* aplikasi yang bersangkutan atau yang merupakan bagian dari Android API. Metode deteksi *malware* yang digunakan adalah dengan meneliti bagian – bagian tertentu dari *functions*, yang dalam hal ini merupakan *hooked functions*, yang dipanggil oleh aplikasi [8]. Setelah itu, sistem akan mengunggah informasi yang berkaitan

dengan koneksi dengan *remote server*. Terdapat 4 komponen utama dari *platform architecture* ini, yaitu *Embedded Client*, Sink pada Smartphone, Web Service, dan *Visualization Component* pada sisi server.

III.1.1.1. Embedded Client

Sistem *Monitoring* terdiri atas 2 bagian, yaitu *embedded client* dan Sink. *Embedded Client* akan dimasukkan ke dalam setiap aplikasi dan dimonitor. Sink akan mengumpulkan kumpulan dari hooked functions yang telah dipanggil oleh aplikasi – aplikasi yang berada dalam pengawasan. *Embedded Client* terdiri atas modul komunikasi yang menggunakan *User Datagram Protocol* (UDP) untuk meneruskan *hooked functions* ke Sink. Pada bagian ini, Javascript Object Notation (JSON) akan digunakan untuk mengirimkan data ke Sink, yang memungkinkan terjadinya pengiriman struktur data dinamis. Untuk mengetahui asal dari *hooked function* yang telah diterima oleh Sink, aplikasi yang berada dalam pengawasan akan menambahkan hash, nama *package*, dan nama aplikasi ke *hooked functions* sebelum mengirimkannya ke Sink. [9] Berikut adalah ilustrasinya :



Gambar 2 Skema Sistem [10]

III.1.1.2. Web Service

Server menyediakan beberapa layanan kepada Sink, antara lain mengunggah aplikasi, mengunduh aplikasi yang sudah dimodifikasi, dan mengirimkan jejak – jejak operasi yang ada. Kunci utama dari keseluruhan sistem adalah *tool* / alat yang digunakan untuk mengimplementasikan aplikasi, yaitu sebuah proses bernama "*hooking*".

Layanan untuk mengunggah file (*file upload service*) memungkinkan Sink untuk mengirimkan target aplikasi yang hendak dimonitor / diawasi dan men-*trigger* instruksi / perintah untuk memasukkan seluruh *hook* yang ada beserta *embedded client*. Selain itu, bagian *web service* ini juga bertugas untuk menyimpan file .apk pada server dan menerima serta mengevaluasi beberapa *permissions* yang terkait. Layanan untuk mengunduh file (*file download*) memungkinkan Sink untuk mengunduh aplikasi yang telah dikirimkan sebelumnya, yang saat ini sedang dalam tahap untuk diawasi / dimonitor. Layanan pemantau jejak (*traces*) memungkinkan Sink untuk mengunggah jejak – jejak yang tersimpan pada perangkat ke *database server* dan menghapus jejak lokal dari SQLite *database*. Setelah *traces* diterima, Web Service akan menyimpannya dalam SQL *database* dan mengirimkan *ack* ke Sink sebagai tanda bahwa *traces* yang dikirimkan berhasil diterima.

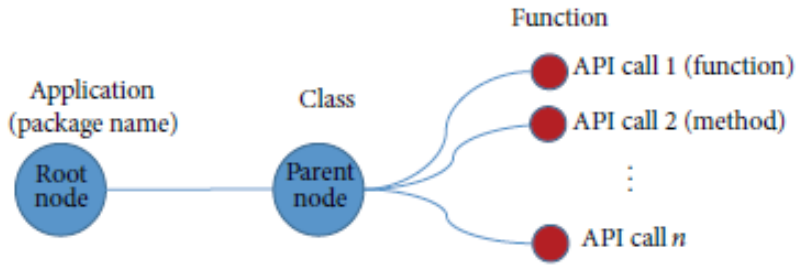
III.2. Analisis *Behavior* dan *Pattern*

Pada tahap ini, dilakukan proses pemasukkan *hooks* pada aplikasi – aplikasi Android yang tersedia. Dilakukan proses *disassembling* dari aplikasi dan akan diteliti kode – kode yang terkandung didalamnya. *Hook Functions* ini dapat meneliti beberapa API *Functions* yang dinilai mencurigakan karena dapat menginjeksi *malware* ke dalam sistem dan menyebabkan kerusakan. Terdapat 6 langkah awal dalam melakukan *hooking*, yaitu sebagai berikut :

a. Menerima aplikasi yang akan di-*hook* dari *smartphone*

b. Melakukan proses *unpacking* dan *disassembling*
c. Memodifikasi file – file aplikasi
d. Melakukan *Packing* terhadap aplikasi yang sudah ter-*hooked* dengan menggunakan Android-apktool
e. Melakukan *Signing* terhadap aplikasi yang sudah ter-*hooked*
f. Mengirimkan *hooked application* setelah *request* diberikan oleh *smartphone*.

Setelah semua proses *hook* tersebut selesai dilakukan, perlu dilakukan visualisasi. Visualisasi ini bertujuan untuk memberikan analisis yang menyeluruh terhadap *behavior* dari aplikasi – aplikasi yang telah masuk dalam pengawasan / *monitoring* pada tahap sebelumnya. Berikut adalah gambaran secara umum mengenai proses ini :



Gambar 3 Visualisasi Skema pemanggilan *API Calls Function* [11]

Root Node berisi nama *package* dari aplikasi, yang bersifat unik untuk setiap aplikasi – aplikasi yang telah ter-*install* di perangkat *smartphone*. *Node* kedua yang berada di tengah, yang bernama *Parent Node*, merepresentasikan nama dari komponen – komponen Android yang telah memanggil API *Call*. Node Ketiga, yang bernama *Function*, merepresentasikan nama – nama dari seluruh fungsi dan metode – metode yang di-*invoked* oleh aplikasi yang bersangkutan. Setiap aplikasi mungkin memiliki beberapa *class* dan setiap *class* memiliki beberapa *functions* dan *methods*.

Setelah itu, akan dilkukan pendalaman dan analisis terhadap *Rules* yang berperan sebagai protokol untuk setiap aplikasi – aplikasi pada Android. Bagian *Rules* ini akan men-*highlight* API *Calls* yang bersifat *restricted*, yang memberikan akses kepada akses ke data – data dan *resources* yang sensitif atau

resources pada *smartphone* yang ditemukan pada sampel – sampel *malware* yang diperoleh berdasarkan penelitian. Pada tahap ini, akan dilakukan analisis dinamis (*dynamic analysis*), dengan melakukan observasi dan penelitian terhadap *Java Based* API *Calls* yang biasanya berperan dalam proses – proses *runtime* pada aplikasi – aplikasi *smartphone* berbasis Android. Adapun peranan yang dilakukan berupa pengaksesan data oleh aplikasi, menentukan lokasi *user*, penulisan data ke *file* output, list panggilan telepon, pengiriman SMS, jalur aliran pengiriman dan penerimaan data dari dan menuju jaringan [12].

Berikut ini adalah contoh API Functions yang berdasarkan penelitian yang dilakukan pada tahap – tahap sebelumnya, memiliki kode – kode *malicious* yang dapat memberikan kerusakan terhadap sistem, sebagai berikut:

a. API *Calls* untuk mengakses data – data sensitif, seperti getSimSerialNumber(), getImei, dan getDeviceId()
b. API *Calls* untuk melakukan komunikasi antar jaringa – jaringan yang tersedia, seperti setWifiEnabled() dan execHTTPRequest()
c. API *Calls* untuk mengirimkan dan menerima *messages*, seperti sendText(), sendBroadcast(), sendAttachment()
d. API *Calls* untuk membocorkan lokasi tempat *user* berada, seperti getLastKnownLocation(), requestLocationUpdates(), dan getCoordinates()
e. API *Function Calls* untuk melakukan eksekusi *command* tertentu, seperti Runtime.exec()
f. API *Calls* yang digunakan untuk proses *obfuscation*, seperti DexClassLoader.loadClass() dan Cipher.getInstance().

Berdasarkan kumpulan *functions* yang *suspicious* tersebut, maka dapat dibuat kumpulan *rules* – *rules* yang mampu mendefinisikan keberadaan *malware* dengan pernyataan – pernyataan kondisional. Contohnya adalah sebagai berikut :

a. Peraturan yang menunjukkan bahwa aplikasi yang diteliti tidak diberikan akses untuk mengetahui lokasi *user* :

- IF Not (ACCESS_User_Location) AND CALL_getLastKnownLocation THEN Malware.

b. Peraturan yang memungkinkan *analyst* untuk mendeteksi bahwa aplikasi yang bersangkutan sedang berusaha untuk mengakses data – data sensitif dari *smartphone* tanpa *Permissions* :

- IF NOT (READ_PHONE_STATE) AND CALL_getImei THEN Malware.

III.3. Hasil Evaluasi

Setelah metode – metode analisis *behavior* dan *pattern* tersebut dilakukan dan seluruh *function* berhasil diidentifikasi, maka sang *analyst* dapat mengumpulkan seluruh *functions* API *Call* yang dinilai *suspicious* atau mencurigakan. Function – function seperti getUserLastKnownLocation, obtainUserId(), getUserInfo() adalah serangkaian contoh API *Call Functions* yang dinilai berbahaya karena *malware* dapat melakukannya tanpa seizin dari Permissions yang menandakan bahwa *malware* tersebut dapat mengambil data dan informasi tersebut tanpa sepengetahuan *user* yang bersangkutan.

IV. Penutup

IV.1. Kesimpulan

Metode Analisis *Behavior* dan *Pattern* dari API *Call Sequence* ini dapat dinilai cukup efektif untuk mendeteksi keberadaan *malware* pada aplikasi yang dapat diunduh oleh *user* pada platform Third Party App yang tersedia dimana – mana. Dengan adanya Hook Functions dan deklarasi Rules yang tepat, maka *Malware Analyst* dapat mengklasifikasi dan mengumpulkan jenis – jenis API *Call Funcitions* apa saja yang dinilai berbahaya.

IV.2. Saran

Untuk mencegah masuknya *malware* ke dalam perangkat *smartphone* berbasis Android, maka *user* perlu memerika kredibilitas aplikasi yang hendak di-*download*. *User* sangat tidak dianjurkan untuk mengunduh aplikasi dari Third Party App yang mencurigakan. Kasus ini seringkali terjadi karena Third Party App sifatnya gratis dan dinilai efisien. Padahal, banyak Third Party App yang telah melakukan injeksi *malicious code* ke dalam aplikasi – aplikasi yang disuguhkan sehingga *user* yang men-*download* aplikasi tersebut akan rentan terhadap pencurian informasi dan mungkin perangkatnya akan mengalami kerusakan dan kekacauan karena *malware* mungkin saja dapat merubah *registry* dari perangkat Android yang bersangkutan.

V. Referensi

[1] Jerbi, Manel & Dagdia, Zaineb & Bechikh, Slim & Makhlouf, Mohamed & Ben Said, Lamjed. (2020). On the Use of Artificial Malicious Patterns for Android Malware Detection. Computers & Security. 92. 101743. 10.1016/j.cose.2020.101743.

[2] Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., Awajan, A. Intelligent mobile malware detection using permission requests and API calls (2020) Future Generation Computer Systems

[3] Alazab, M., Layton, R., Venkataraman, S., & Watters, P. (2010). Malware detection based on structural and behavioural features of api calls.

[4] Patanaik, Chinmaya & Barbhuiya, Ferdous & Nandi, Sukumar. (2012). Obfuscated malware detection using API call dependency. ACM International Conference Proceeding Series. 185-193. 10.1145/2490428.2490454.

[5] Alazab, M., Layton, R., Venkataraman, S., & Watters, P. (2010). Malware detection based on structural and behavioural features of api calls.

[6] Alazab, M., Layton, R., Venkataraman, S., & Watters, P. (2010). Malware detection based on structural and behavioural features of api calls.

[7] Alazab, M., Layton, R., Venkataraman, S., & Watters, P. (2010). Malware detection based on structural and behavioural features of api calls.

[8] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, *2016*.

[9] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, *2016*.

[10] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, *2016*.

[11] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, *2016*.

[12] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, *2016*.