# BAB II

# KAJIAN PUSTAKA

## A. *ANDROID*

Menurut Sharma (2014) *Android* adalah sistem operasi yang berbasis linux untuk telepon seluler seperti smartphone dan tablet PC. *Android* menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang akan digunakan oleh bermacam perangkat seluler. Awalnya, *Google Inc* membeli *android* Inc, pendatang baru yang membuat perangkat lunak untuk ponsel. Kemudian untuk mengembangkan *android*, dibentuklah *Open Handset Alliance*, *konsorsium* dari 34 perusahaan perangkat keras, perangkat lunak, dan telekomunikasi, termasuk *Google, HTC, Intel, Motorola, Qualcomm, T-Mobile dan Nvidia.*

Pengertian *Android* menurut Supardi (2011) adalah sebuah sistem operasi perangkat mobile berbasis linux yang mencakup sistem operasi, *middleware*, dan aplikasi. Beberapa pengertian lain dari *Android* , yaitu:

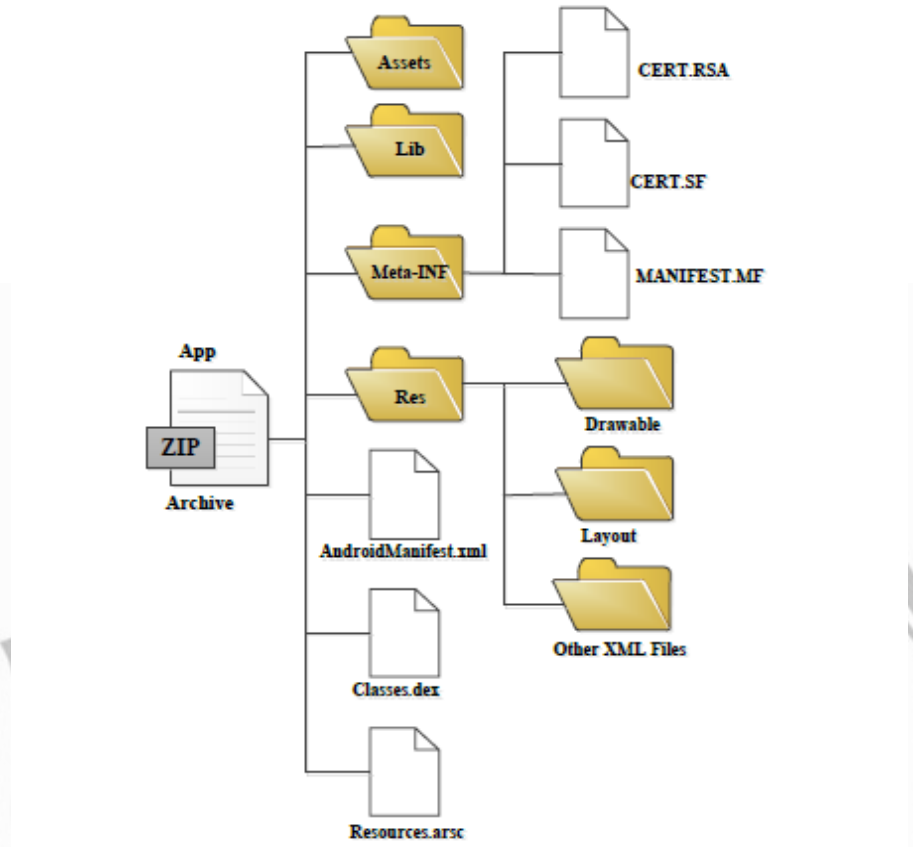
1. Merupakan platform terbuka (*Open Source*) bagi para pengembang (*programmer*) untuk membuat aplikasi.
2. Merupakan sistem operasi yang dibeli *Google Inc.* dari *Android* Inc.
3. Bukan bahasa pemrograman, akan tetapi hanya menyediakan lingkungan hidup atau run time environment yang disebut DVM (*Dalvik Virtual*

*Machine*) yang telah dioptimasi untuk *device*/alat dengan sistem memori yang kecil.

## B. STRUKTUR FILE *ANDROID PACKAGE*

Paket aplikasi *android* memiliki ekstensi file *.apk (*Android Package Kit*). File apk merupakan bentuk terkompresi dari paket-paket file dalam aplikasi. Aplikasi *Android* berisi file dan folder berikut (Tiwari dkk, 2016).

1. File *Android Manifest.xml* : file ini merupakan dengan format xml paling penting dalam struktur aplikasi *android* karena berisi informasi konfigurasi aplikasi seperti versi *android* dan *permission* yang dibutuhkan aplikasi.
2. Folder META-INF : folder ini berisi file CERT.RSA, CERT.SF, dan MANIFEST.MF. Folder ini berisi informasi *signature* dari setiap file dalam file apk aplikasi.
3. File *Classes.dex* : file ini merupakan *bytecode* aplikasi yang telah di *compile* dengan *Java VM* sehingga bisa diterjemahkan oleh *dalvik VM*.
4. Folder res : folder ini berisi file-file yang dibutuhkan oleh aplikasi seperti *layout, atribut, icon* dan lain-lain.
5. File *resources.asrc* : file *binary resource* yang dihasilkan setelah kompilasi berhasil dilakukan.
6. Folder *Assets* : folder berisi file tambahan untuk aplikasi.
7. Folder *lib* : folder berisi *private library* untuk aplikasi.




Gambar 1. Komponen didalam file apk Aplikasi *Android* (Faruri dkk, 2015)

## C. *MALWARE*

*Malicous Software* yang biasa dikenal dengan sebutan *malware* merupakan sebuah aplikasi yang dirancang khusus untuk dapat menyusup kedalam sistem tanpa diketahui pemilik sistem. Aplikasi tersebut umumnya memuat sebuah perintah yang telah dibuat dengan tujuan khusus. Perintah tersebut seperti menyebarkan *virus, Trojan, Worm*, atau memasang *backdoor* didalam sistem. Lebih sederhananya *Malware* merupakan sebuah aplikasi yang dirancang untuk membuat celah pada keamanan sistem komputer. (Sikroski, 2012)

*Malware* menginfeksi ponsel pintar melalui jalur-jalur yang tidak resmi. Toko aplikasi pihak ke-3 maupun mengunduhnya di situs web tidak jelas adalah

jalan bagi *malware* menginfeksi ponsel pintar. Karena *malware* dapat mengeksploitasi perangkat yang terinfeksi dengan cara menyisipkan barisan *code malware* pada suatu aplikasi yang mirip dengan aplikasi yang terdapat di *official market* seperti *playstore,* lalu pelaku mengunggahnya kembali pada toko aplikasi pihak ke-3 sebagai perantara sehingga kejahatan ini sangat penting untuk diselidiki.

Menurut Friska (2017) dalam penelitiannya tentang taksonomi serangan pada *android*, salah satu sumber *threat* pada serangan pada *android* dapat berasal dari *malware* dengan target serangan yaitu aplikasi *android*. Sehingga perlu dilakukan penelitian untuk mendeteksi keberadaan *malware* didalam aplikasi.

Ada 2 pendekatan dalam melakukan analisa *malware*: statis dan dinamis. Statis lebih kepada pendekatan struktur *malware* itu dibuat, dengan kata lain melakukan *reverse engineering* pada *malware* tersebut. Sedangkan dinamis lebih kepada pendekatan analisa yang secara kasat mata atau dengan kata lain interaksi penuh dengan *malware* dengan menjalankan file *malware* tersebut pada lingkungan virtual. (Sikroski, dkk 2012)

*Malware analysis* adalah proses untuk melakukan analisis *malware* dan cara mempelajari komponen dan perilaku *malware*. Pada penelitian ini akan menggunakan dua metode analisis *malware*, analisis statis dan analisis dinamis. Analisis statis adalah metode analisis *malware* yang dilakukan tanpa menjalankan *malware*. Sedangkan analisis dinamis adalah metode analisis *malware* yang dijalankan *malware* dalam sistem aman (Sikroski dkk, 2012).

## D. METODE INFEKSI *MALWARE*

Beberapa metode *malware* penyebarkan infeksi *malware* ke dalam *handphone* adalah sebagai berikut (Zou dan Jian, 2012).

1. *Repackaging* : *attacker* menggunakan aplikasi asli dan mengganti atau menyisipkan kode berbahaya didalam aplikasi tersebut, ketika korban sudah terinfeksi maka *attacker* bisa menguasai *handphone* secara *remote* tanpa izin *user* dan bahkan dapat melakukan akses informasi pribadi *user*.
2. *Update attack* : aplikasi pertama biasanya hanya sebagai pengecoh, karena tidak terdapat fungsi mencurigakan, namun aplikasi memiliki fungsi *update* dari sumber yang tidak resmi untuk *update* aplikasi tersebut.
3. *Drive by download* : menarik *user* agar mengunduh suatu aplikasi melalui *browser*. *GGtracker, Jifake, Spitmo dan ZitMo* adalah contoh *malware* yang menginfeksi *user* dengan metode *drive by download*.
4. *Misusing Android 's application bug* : menggunakan kerentanan aplikasi yang ada untuk menginfeksi *malware* kedalam sistem *Android* .

   Beberapa metode lain merupakan turunan dari metode diatas seperti *fake application* dan *remote install*.

## E. *MALWARE ANALYSIS*

*Malware analysis* adalah proses untuk melakukan analisis *malware* dan cara mempelajari komponen dan perilaku *malware*. Pada penelitian ini akan menggunakan dua metode analisis *malware*, analisis statis dan analisis dinamis. (Sikroski dkk, 2012).

Metode pertahanan terhadap *malware* pada platform *Android* dibagi menjadi 2 (Raveendranath dkk, 2014), yaitu:

1. ***Static Analysis*** yaitu metode deteksi *malware* tanpa menjalankan aplikasi. Teknik ini biasa disebut *code exploration* karena menganalisis kode dari *malware*. Berdasarkan variasi teknik transformasi kode *static analysis* dibagi 5 kategori (faruki dkk, 2015), yaitu.
   a. *Signature-Based Malware Detection* : *Anti-Malware* saat ini bekerja dengan menggunakan metode *Signature-Based Malware Detection* karena metode ini sangat efisien untuk menghadapi *malware* yang sudah dikenali. Dengan mengekstrak kode *malware* dan membuat *unique signature* yang cocok dengan bagian *malware* tersebut. Namun si*gnature-base* tidak mampu mendeteksi jenis *malware* baru yang belum pernah diketahui *signature* nya.
   b. *Component-Based Analysis :* untuk melakukan analisa aplikasi dengan lebih detail dan terperinci, aplikasi yang dicurigai *malware* dapat di ekstrak untuk mendapatkan konten penting seperti file *Android Manifest.xml, resources* dan *bytecode.* File *Manifest* menyimpan *metadata* penting tentang daftar komponen aplikasi (seperti *activities, services, receivers*, dan lain-lain) dan juga *permission* yang diperlukan aplikasi. Analisis komponen dan interaksi *bytecode* untuk mengidentifikasi kerentanan dari aplikasi tersebut.
   c. *Permission-Based Analysis* : akses *permission* untuk *resource* penting adalah desain utama bagi model security *android*. Identifikasi *permission* berbahaya tidak cukup untuk menyatakan aplikasi mengandung *malware*. Namun memetakan *permission* yang diminta dan *permission* yang

digunakan merupakan teknik identifikasi yang penting untuk dilakukan (Barrera dkk, 2012).

d. *Dalvik Bytecode Analysis* : *Dalvik bytecode* menyimpan informasi *class*, metode dan instruksi aplikasi sehingga dapat digunakan untuk verifikasi perilaku aplikasi. Analisis mendalam mengenai *control* dan *data flow* dapat menunjukan fungsi berbahaya dari aplikasi seperti *leakage privacy* dan penyalahgunaan layanan *handphone* (Faruki dkk, 2015).

e. *Re-targeting Dalvik Bytecode to Java Bytecode* : *availability* jumlah *Java decompiler* dan *tool-based* analisis statis memotivasi para peneliti untuk meneliti kembali *dalvik bytecode* lebih dalam dengan mengubah menjadi *source java*. Sebagai hasilnya saat ini berbagai macam *tool* telah ada untuk konversi *dalvik bytecode* menjadi *Java bytecode*. Selanjutnya mereka akan melakukan analisis dan identifikasi *malware* (Faruki dkk, 2015).

2. ***Dynamic Analysis*** yaitu analisis *malware* dengan menjalankan aplikasi didalam lingkungan terlindungi dan disediakan semua *resource* yang dibutuhkan sehingga aktifitas dari *malware* dapat dipelajari. Teknik *dynamic analysis* perlu dilakukan untuk menutupi kelemahan dari *static analysis* yaitu gagal mendeteksi *malware* yang dienkripsi, *polymorphic malware*, dan *code transform malware* lainnya. Sedangkan kelebihan dari *static analysis* yaitu dapat mendeteksi keberadaan *malware* dengan cepat. *Dynamic analysis* dibagi menjadi 3 kategori (Faruki dkk, 2015), yaitu.

a. *Profile-based Anomaly Detection* : mendeteksi *malware* dengan monitoring penggunaan *hardware*. Rentang parameter seperti penggunaan *CPU,*

*statistic* penggunaan *RAM*, penggunaan jaringan, penggunaan baterai, *system-calls* dipelajari untuk mendeteksi perilaku abnormal dari aplikasi.

b. *Malicious Behaviour Detection*: mendeteksi keberadaan *malware* berdasarkan perilaku berbahaya seperti *data leakage, sending SMS/email, voice call* tanpa persetujuan dari *user*.

c. *Virtual Machine Introspection* : metode ini untuk menutupikelemahan *behavior detection* dengan *emulator*, karena ada kemungkinan *emulator* juga telah dikalahkan oleh *malware*. Untuk mengatasi hal ini, *Virtual Machine Introspection* dapat digunakan yaitu dengan mengamati perilaku emulator untuk mendeteksi perilaku aplikasi.

## F. *REVERSE ENGINEERING*

Secara umum pengertian *Reverse Engineering* adalah proses dari ekstraksi pengetahuan atau *blue-print design* dari apapun yang telah dibuat manusia. Konsep *Reverse Engineering* sudah ada sejak sebelum teknologi modern atau komputer dibuat. *Reverse Engineering* biasa digunakan oleh industri untuk mengetahui suatu informasi dari proses, desain, atau ide yang dibuat sebelumnya namun memiliki sedikit informasi untuk dikembangkan lebih lanjut. [Eilam 2007]. Di dalam dunia tekologi informasi *Reverse Engineering* berhubungan erat dengan *Software* atau Aplikasi, istilah lain yang dikenal dari *Reverse Engineering* adalah *Reverse Code Engineering*.

Proses *Reverse Engineering* pada sebuah software atau aplikasi dapat dilakukan dengan cara :

1. ***Assembly.*** *Assembly language* merupakan bahasa pemrograman yang berada pada level rendah dari beberapa bahasa pemrograman yang dikenal selama ini. Bahasa *assembly* digunakan untuk sebuah mesin karena mesin tidak dapat mengenal bahasa pemrograman tingkat tinggi seperti *java, basic, pascal*, dll.
2. ***Disassembly.*** *Disassembly* merupakan kebalikan dari proses *assembly*. Proses *disassembly* digunakan dalam teknik *Reverse Engineering* untuk menerjemahkan dari Bahasa mesin ke bahasa yang mudah dimengerti manusia, yaitu bahasa *assembly*.
3. ***Hashing.*** *Hash* merupakan identitas dari sebuah program seperti halnya sidik jari pada manusia. Proses *hash* dilakukan untuk verifikasi sebelum dan setelah proses analisa *malware*. Verifikasi tersebut dilakukan untuk mengetahui tidak adanya perubahan *hash* pada *sample malware* setelah dilakukan proses analisis.
4. ***String Analysis.*** *String* atau karakter dalam sebuah program yang merupakan nilai yang akan dilakukan proses *load* oleh sample *malware* ketika dieksekusi. Hal ini yang menjadikan dalam proses *reverse engineering* harus dilakukan *string* analisis untuk mendapatkan bukti kuat dari sample *malware*.
5. **MAER (*Malware Analysis Environment and Requirement*).** MAER adalah ruang lingkup yang menjadi laboratorium analisis *malware*. MAER merupakan salah satu penentu seorang analis *malware* mendapatkan infromasi yang akurat dan efisien dari analisa yang dilakukan.
6. ***Repository Malware*.** Repository *malware* merupakan tempat disimpannya sampel *malware* yang telah berhasil melakukan serangan kedalam sistem

komputer dimanapun. *Repository malware* dibuat untuk memberikan sampel kepada seorang *malware analyst* untuk melakukan analisa terhadap *malware* yang sudah berhasil melakukan serangan. (*virusshare*). *Repository Malware* adalah salah satu dari *malware source* yang dapat digunakan untuk kepentingan analisis.

## G. KAJIAN PENELITIAN TERDAHULU

1. Prayudi, dkk. (2015) melakukan analisa *malware* dengan menggunakan metode statis dan dinamis pada perangkat PC. Dalam penelitian ini, mereka dapat menemukan waktu penciptaan *malware*. Dan dengan analisis statik canggih mereka mampu menemukan informasi lengkap tentang karakteristik *malware*. Sedangkan dalam metode dinamis mereka dapat menemukan bahwa *malware* dapat mematikan sistem keamanan *windows*.
2. Nugroho dan Prayudi (2015) meneliti bagaimana cara kerja *malware Biscuit* yang dilakukan dengan proses *reverse engineering*. Bahwa *malware* jenis *Biscuit* mengirimkan *auto request* pada koneksi IP tertentu, yaitu pada alamat IP 114.101.115.115. lalu dengan proses *reverse engineering* ditemukan perintah: *bdkzt, ckzjqk, download, exe, exit* dan lists pada program.
3. Vijayendra (2016) menemukan penggunaan izin, pemanggilan API dari aplikasi *android* untuk mendeteksi *malware* berbahaya pada platform berbasis *android*. Dalam penelitiannya, dapat ditemukan bahwa menggunakan informasi kategori dengan izin dan panggilan API efektif untuk *malware* deteksi yang mencapai rata-rata 93% deteksi *malware*.

Namun ia tidak melibatkan pelacakan dinamis sehingga tidak ditemukan bagaimana data hasil dari penggunaan teknik dinamis pada penelitiannya.

4. Kunang (2014) menjelaskan analisa aplikasi *icalender* pada *android* yang ternyata mengandung *malware Trojan* yang dapat mengirimkan sms pada *premium number* tertentu. Lalu pada aplikasi *live prints wallpaper* yang ternyata mengandung *malware* jenis *Trojan* dan *spyware* dengan menggunakan teknik forensik. Namun tidak dilakukan analisa dinamis sehingga tidak ditemukan proses pengiriman data pada *traffic network* .
5. Weichselbaum, dkk (2014) melakukan analisa *malware* dengan menggunakan ANDRUBIS, *framework* analisis yang sepenuhnya otomatis menganalisis sistem skala besar untuk aplikasi *android* yang menggabungkan teknik analisis statis dengan analisis dinamis. Mereka menerapkan beberapa teknik stimulasi untuk memicu perilaku selama analisis dan memverifikasi keefektifannya dengan mengevaluasi cakupan kode yang dihasilkan.