JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER

USULAN TUGAS AKHIR

1. IDENTITAS PENGUSUL

NAMA	**: Yolanda Septiana Dewi**
NRP	**: 5109100187**
DOSEN WALI	**: Dwi Sunaryono, S.Kom., M.Kom.**
DOSEN PEMBIMBING	**: 1. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc.**
	2. Baskoro Adi Pratomo, S.Kom., M.Kom.

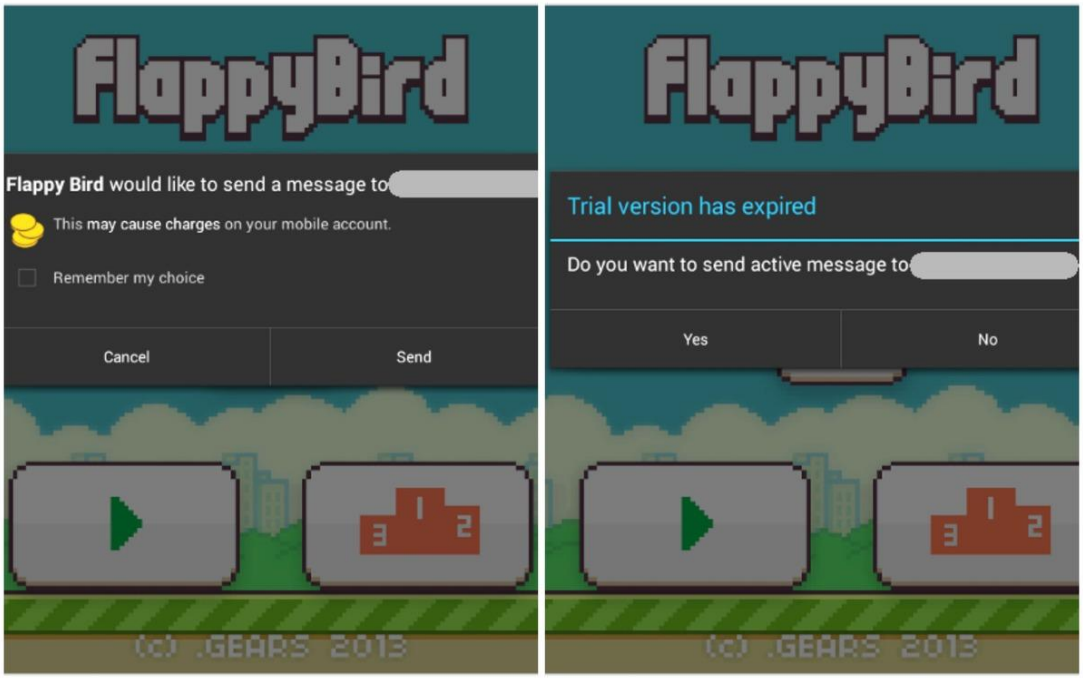
2. JUDUL TUGAS AKHIR

"Rancang Bangun Sistem Pendeteksi Kemiripan Aplikasi pada Android"

3. LATAR BELAKANG

Cybercriminal dapat dengan mudah me-*repackage* aplikasi Android karena aplikasi Android yang berekstensi .apk pada dasarnya hanyalah sebuah *file* ZIP dan banyak perangkat telah tersedia untuk melakukan *reverse engineering* seperti **apktool** dan **dex2jar**. Tujuan *cybercriminal* untuk me-*repackage* sebuah aplikasi antara lain adalah sebagai tindakan plagiarisme ataupun penyebaran *malware*. Aplikasi terkenal, terutama dalam kategori *game*, seringkali menjadi sasaran *repackaging* dan kemungkinan memuat *malware* berbahaya seperti SMS premium, pencurian identitas, perusak *file*, dan lain-lain.

Pada bulan Februari 2014, *game mobile* Flappy Bird ditarik dari pasaran oleh pembuatnya karena *game* tersebut terlalu adiktif. Tak lama kemudian puluhan *game clone* atau *repackage* pun bermunculan dan beberapa di antaranya mengandung *malware* SMS premium [1]. *Screenshot* dari *game* Flappy Bird palsu ada di Gambar 1.



Gambar 1. Versi *malicious* dari Flappy Bird yang akan menguras pulsa penggunanya melalui SMS premium.

Dari penelitian yang dilakukan oleh **Android Malware Genome**, sebanyak 1083 dari 1260 (86%) aplikasi *malware* yang ditemukan merupakan versi *repackage* dari aplikasi sah yang kemudian ditambahkan *payload malware* oleh *developer* jahat [2]. Aplikasi *repackage* tersebut biasanya beredar di pasar *third party* dan menawarkan aksesibilitas yang tidak bisa didapatkan di pasar resmi seperti Google Play. Aksesibilitas yang ditawarkan oleh aplikasi *malware* tersebut misalnya adalah dapat diunduh secara gratis meskipun sebenarnya aplikasi yang sah merupakan aplikasi berbayar atau tidak tersedia di wilayah tertentu.

Oleh karena faktor-faktor tersebut, sebuah teknik untuk mendeteksi aplikasi *repackage* Android akan dibuat sebagai tugas akhir ini menggunakan metode ***Program Dependence Graph* (PDG)**. Sistem pendeteksi akan membandingkan satu aplikasi sah dengan banyak aplikasi lainnya kemudian menampilkan hasil kemiripan. Meskipun aplikasi sah dan aplikasi *repackage* mungkin mengandung perbedaan, metode deteksi *repackaging* menggunakan *Program Dependence Graph (PDG)* telah terbukti efektif menangani teknik modifikasi seperti penataan kembali *statement*, penambahan, dan penghapusan kode. Walaupun telah dimodifikasi, sebagian besar fungsionalitas aplikasi *repackage* tetap sama dengan aplikasi sah sehingga tetap dapat terdeteksi menggunakan metode PDG.

4. RUMUSAN MASALAH

Rumusan permasalahan yang akan diselesaikan pada tugas akhir ini adalah sebagai berikut:

1) Merepresentasikan *Java bytecode* ke dalam *Program Dependence Graph*.
2) Melakukan *preprocessing* terhadap *Program Dependence Graph* (PDG) untuk mengurangi jumlah pasangan PDG yang akan dibandingkan.
3) Menghitung nilai isomorfisme pada pasangan *Program Dependence Graph*.
4) Menghitung nilai kemiripan pada dua aplikasi.

5. BATASAN MASALAH

Permasalahan yang akan diselesaikan dalam tugas akhir ini memiliki batasan yaitu:

1) Tidak melakukan analisis pada kode *native* yang ditulis menggunakan bahasa C atau C++ namun hanya memeriksa *byte DEX* (*Dalvik Executable*) yang ditulis menggunakan Java.

6. TUJUAN TUGAS AKHIR

Tujuan dibuatnya tugas akhir ini adalah untuk:

1) Mendeteksi aplikasi Android apakah merupakan hasil *repackage* atau bukan dengan menghitung nilai isomorfisme *Program Dependence Graph* dari aplikasi tersebut dan aplikasi sah.

7. MANFAAT TUGAS AKHIR

Manfaat dari tugas akhir ini adalah dapat mendeteksi aplikasi Android yang merupakan hasil *repackaging* dari aplikasi sah sehingga mengurangi kemungkinan tersebarnya *malware*.

8. TINJAUAN PUSTAKA

a) Pengertian *repackage*

Sebuah aplikasi disebut *repackage* apabila aplikasi tersebut (1) mempunyai kode yang serupa dengan aplikasi sah namun (2) mempunyai kepemilikan yang berbeda. Oleh sebab kriteria kedua, maka *library* eksternal harus disisihkan dari sistem deteksi PDG karena *library* eksternal ditujukan untuk digunakan kembali oleh *developer* lain. Selain itu, aplikasi yang terdeteksi sebagai hasil *repackage* namun mempunyai kepemilikan sama dengan aplikasi sah tidak akan dianggap sebagai hasil *repackage*.
Aplikasi *repackage* juga disebut dengan *clone*, berikut tipe-tipenya [3]:

Tipe 1: Memiliki fragmen kode yang identik kecuali variasi pada *whitespace*, *layout*, dan komentar.

Tipe 2: Memiliki fragmen kode yang identik secara sintaksis kecuali variasi pada *identifier*, *literal*, tipe, *whitespace*, *layout*, dan komentar.

Tipe 3: Memiliki fragmen kode yang disalin dengan modifikasi seperti perubahan, penambahan, pengurangan *statement*, dan variasi pada *identifier*, literal, tipe, *whitespace*, *layout*, dan komentar.

Tipe 4:Memiliki fragmen kode yang melakukan komputasi yang sama namun diimplementasikan dengan sintaksis yang berbeda.

b) Analisis statis dan analisis dinamis

Analisis statis merupakan teknik analisis yang berusaha untuk memeriksa baris kode dengan cara pemindaian dan tanpa menjalankan aplikasi tersebut. Sebaliknya, analisis dinamis berusaha untuk melakukan proses analisa di saat aplikasi berjalan dengan memeriksa aliran sistem satu per satu. Analisis dinamis lebih sulit untuk dilakukan karena memerlukan otomasi *input* yang cukup kompleks. Dalam tugas akhir ini dilakukan teknik analisis statis.

c) Struktur aplikasi Android

File aplikasi Android berformat .apk yang pada dasarnya adalah *file* .zip yang terdiri dari *file-file* yang telah disusun menjadi satu, terdiri dari:

- Direktori `META-INF`, yang berisi *file* `MANIFEST.MF`, `CERT.RSA`, `CERF.SF`.
- Direktori `lib`, berisi direktori-direktori yang mengandung kode yang di-*compile* khusus untuk prosesor tertentu, misalnya:
  - Direktori `armeabi` yang berisi kode yang telah di-*compile* untuk prosesor berbasis ARM.
  - Direktori `armeabi-v7a` yang berisi kode yang telah di-*compile* untuk prosesor berbasis ARMv7 dan kelanjutannya.
  - Direktori `x86`.
  - Direktori `mips`.
- Direktori `res`, berisi *file-file* aset yang tidak di-*compile*, seperti gambar.
- Direktori `assets`, berisi *file-file* aset yang dibuka melalui `AssetManager`.
- *File* `AndroidManifest.xml`. Setiap aplikasi Android pasti memiliki *file* ini yang berisi informasi tentang versi aplikasi, nama aplikasi, komponen apa saja yang dimiliki, *permission* apa saja yang diperlukan. *File* ini dalam keadaan ter-*compile* sehingga tidak terbaca oleh manusia. Untuk membukanya diperlukan perangkat seperti Androguard atau apktool.
- *File* `classes.dex`, kumpulan semua kode yang ditulis oleh *developer*, termasuk *library*, yang di-*compile* dalam bentuk Dalvik Executable (DEX).
- *File* `resource.arsc`, berisi *resource* yang di-*compile* seperti *file* XML.

d) Program Dependence Graph

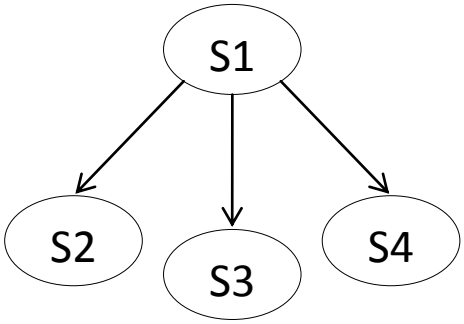
Program Dependence Graph (PDG) merupakan representasi dari sebuah program dimana setiap *node* adalah *statement* dan *edge* menunjukkan hubungan ketergantungan data atau kontrol antar *statement*. Sebuah PDG terdiri dari *Data Dependence Graph* dan *Control Flow Graph*.

Terdapat dua jenis dependensi pada PDG:

a) Dependensi data
Dependensi data menggambarkan hubungan ketergantungan data antar *statement*. Contohnya jika terdapat sebuah *statement* `S2` yang memiliki variabel `t1` yang diproses terlebih dahulu di *statement* `S1`, maka `S1` memiliki *edge* menuju `S2`. Dependensi data bisa direpresentasikan ke dalam *Data Dependence Graph*. Contoh *pseudocode* yang menunjukkan dependensi data terdapat pada Gambar 2 dan Gambar 3 menunjukkan representasi graf-nya.

```
S1.  t1 := ld(x);
S2.  t2 := t1 + 4;
S3.  t3 := t1 * 8;
S4.  t4 := t1 – 4;
```

Gambar 2. *Pseudocode* program sederhana yang menunjukkan dependensi data.

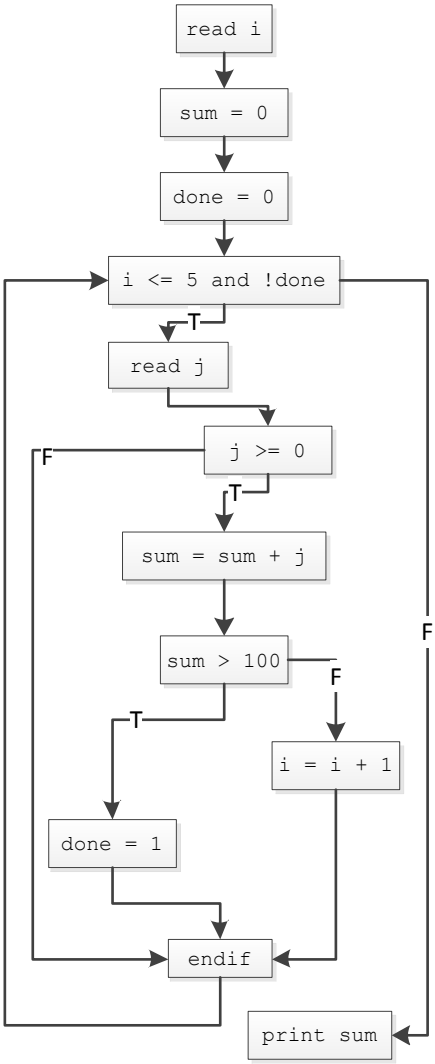


Gambar 3. *Data Dependence Graph* dari Gambar 2.

b) Dependensi kontrol
Dependensi kontrol terjadi apabila pengeksekusian sebuah *statement* `S1` bergantung ke sebuah kondisi, misalnya apakah nilai `A` memenuhi suatu kriteria. Dependensi kontrol biasanya direpresentasikan dengan *Control Flow Graph*, yaitu sebuah graf berarah `G` yang mempunyai titik awal `START` dan titik akhir `STOP` dan setiap node `N` memiliki paling banyak dua *successor*. Untuk setiap node `N` pada `G`, terdapat *path* dari `START` ke `N` dan dari `N` ke `STOP`. Contoh program sederhana dan representasi *Control Flow Graph* ada pada Gambar 4 dan Gambar 5.

```
S1.  read i
S2.  sum = 0
S3.  done = 0
P4.  while i <= 5 and !done do
S5.    read j
P6.    if j >= 0 then
S7.      sum = sum + j
P8.      if sum > 100 then
S9.        done = 1
       else
S10.       i = i + 1
       endif
     endwhile
S11. print sum
```

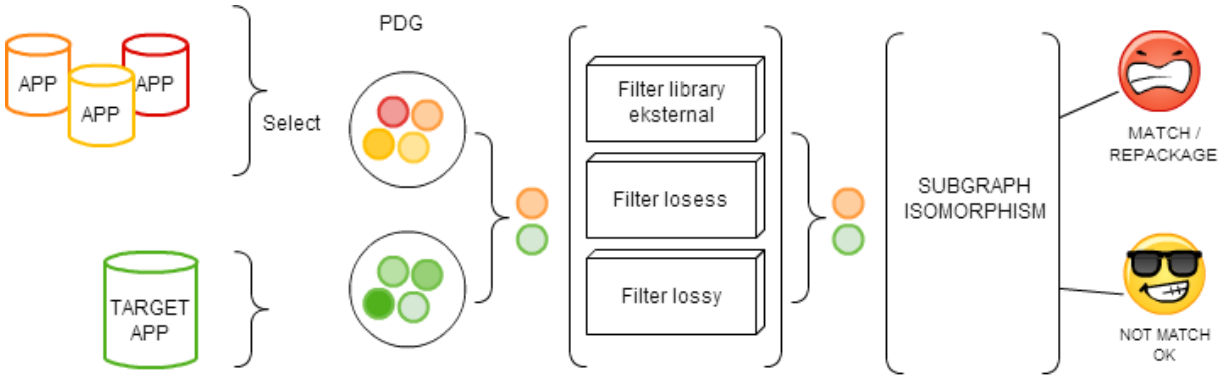
Gambar 4. *Pseudocode* program sederhana yang menunjukkan dependensi kontrol.



Gambar 5. *Control Flow Graph* dari Gambar 4.

9. RINGKASAN TUGAS AKHIR

Metodologi yang akan dilakukan pada tugas akhir ini adalah sesuai dengan Gambar 6.



Gambar 6. Metodologi tugas akhir.

a) Mendapat aplikasi sah

Aplikasi sah telah disiapkan sebelumnya dengan mengambil dari Google Play dan akan dibandingkan dengan aplikasi-aplikasi yang ada di *file storage*. Tempat penyimpanan *file* (*file storage*) adalah sebuah *disk* yang digunakan untuk menyimpan semua aplikasi yang akan dibandingkan dengan aplikasi sah.

Sistem pendeteksi harus membandingkan semua kemungkinan pasangan PDG yang ada, oleh karena itu harus dilakukan *preprocessing* (filter) terlebih dahulu untuk mereduksi jumlah pasangan PDG yang harus dibandingkan. Filter yang dilakukan adalah penghilangan *library* eksternal, filter *lossless*, dan filter *lossy*.

b) Mengekstrak *file* DEX dari APK

Kedua *file* APK (aplikasi sah dan aplikasi dari *file storage*) akan diekstrak untuk diambil sebuah *file* yang berekstensi *.dex*. *File* DEX adalah *file* yang berisi *class-class* yang telah di-*compile* ke dalam format *Dalvik Executable* (DEX). *File* ini kemudian diubah menjadi *Java bytecode* dalam fomat JAR menggunakan dex2jar.

c) Membangun PDG

File DEX akan dibaca lalu dibuat representasi PDG-nya menggunakan WALA [4].

d) Membandingkan PDG

1) Menyaring *library* eksternal

Banyak aplikasi yang memakai *library third-party* seperti Admob atau Facebook API. Oleh karena *library* tersebut tidak dibuat oleh si pemilik aplikasi, maka *library* tersebut harus dikecualikan pada tahap selanjutnya.

2) Filter *lossless*
   PDG yang mengandung kurang dari 10 *node* akan disingkirkan karena *method* berukuran kecil biasanya merupakan fragmen kode yang sudah umum digunakan.
3) Filter *lossy*
   Filter ini berusaha untuk menghilangkan pasangan PDG yang kurang mungkin untuk mempunyai kemiripan dikarenakan perbedaan distribusi tipe pada kedua PDG. Sebagai contoh, sebuah PDG yang memiliki banyak *node* pemanggilan *method* memiliki kemungkinan kecil untuk mirip dengan sebuah PDG yang tidak memiliki *node-node* tersebut. Dalam filter ini akan digunakan tes hipotesa G-test.
4) Isomorfisme subgraf
   Pada tahap ini telah didapatkan pasangan PDG yang akan dihitung tingkat kemiripannya menggunakan algoritma VF2.
5) Menghitung nilai kemiripan
   Untuk setiap *method* f pada sebuah aplikasi A, dimisalkan $|f|$ merupakan jumlah *node* pada sebuah PDG. Jumlah PDG yang serupa dari A dan B dinotasikan sebagai *m*(*f*). Nilai similaritas adalah rasio antara jumlah $|f|$ dan $|m(f)|$.

$$sim_{A(B)} = \frac{\sum_{f \in A} |m(f)|}{\sum_{f \in A} |f|} \quad (1)$$

Persamaan 1 menunjukkan porsi dari aplikasi A yang mempunyai kesamaan pada aplikasi B. Dua aplikasi dikatakan sama apabila salah satu aplikasi mempunyai nilai kesamaan di atas 70% ($\max(sim_{A(B)}, sim_{B(A)}) \geq 70\%$).

10. METODOLOGI

Metodologi yang akan dilakukan secara garis besar adalah sebagai berikut:

1) Penyusunan proposal tugas akhir
   Pada tahap ini akan dilakukan penyusunan proposal tugas akhir yang merupakan langkah awal dalam mengerjakan tugas akhir ini. Proposal mengusulkan tentang pembuatan sistem untuk mendeteksi *repackaging* pada aplikasi Android.
2) Analisis kebutuhan dan studi literatur
   Tahap ini berisi kegiatan pengumpulan informasi mengenai *Program Dependence Graph*, pembelajaran algoritma VF2 dan pencarian cara efisien untuk menyaring *library* eksternal dari sebuah aplikasi.
3) Perancangan sistem
   Pada tahap ini akan dilakukan perancangan dari hasil analisa kebutuhan di tahap 2 dengan membuat konsep solusi dari permasalahan yang sedang dihadapi.
4) Implementasi
   Pada tahap ini dilakukan pembuatan perangkat lunak berbasis *desktop* berdasarkan hasil perancangan di tahap 3. Perangkat lunak dibuat menggunakan bahasa Java serta *library* dex2jar dan WALA. IDE yang akan digunakan adalah Netbeans.
5) Pengujian dan evaluasi
   Pengujian dilakukan dengan menjalankan aplikasi dan mencatat hasilnya. Kemudian evaluasi dilakukan secara manual untuk mengkonfirmasi hasil deteksi dan mengetahui kemungkinan terjadinya *false positive*.
6) Penyusunan buku tugas akhir
   Tahap ini merupakan tahap terakhir dari proses ini. Penulisan buku tugas akhir memiliki sistematika secara garis besar sebagai berikut:
   I. Pendahuluan
      a. Latar belakang permasalahan
      b. Batasan
      c. Tujuan
      d. Metodologi
   II. Tinjauan pustaka
   III. Desain dan implementasi
   IV. Uji coba dan evaluasi
   V. Kesimpulan dan saran
   VI. Daftar pustaka

11. JADWAL KEGIATAN

Jadwal kegiatan pada tugas akhir ini digambarkan pada Tabel 1.

Tabel 1: Rencana pelaksanaan tugas akhir.

Tahapan	2014																			
	Februari				Maret				April				Mei					Juni		
Penyusunan proposal																				
Studi literatur																				
Perancangan sistem																				
Implementasi																				
Pengujian dan evaluasi																				
Penyusunan buku																				

12. DAFTAR PUSTAKA

[1] K. Bel. (2014, Feb.) Mashable. [Online]. http://mashable.com/2014/02/12/flappy-bird-malware/

[2] X. Jiang and Y. Zhou, *Android Malware*. Raleigh, USA/North Carolina: Springer, 2013.

[3] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," Queen's University; University of Bremen, 2009.

[4] J. Dolby, M. Sridharan, and S. Fink. (2014, Feb.) T.J. Watson Libraries for Analysis (WALA). [Online]. http://wala.sourceforge.net/wiki/index.php/Main_Page

[5] J. Crussell, C. Gibler, and H. Chen, "Attack of the Clones: Detecting Cloned Applications on Android Market," Sandia National Labs, 2012.