Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202184500
Vol. 8, No. 4, Agustus 2021, hlm. 831-838 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

SOFTWARE WATERMARKING DINAMIS DENGAN ALGORITME COLLBERG-THOMBORSON DAN PARENT POINTER GRAF PADA APLIKASI ANDROID

Togu Novriansyah Turnip[1], William Suarez Lumbantobing[2], David Christian Sitorus[3], Friska Laurenzia Sianturi*[4]

[1,2,3,4]Fakultas Informatika dan Teknik Elektro, Institut Teknologi Del
Email: Email: [1]togu@del.ac.id, [2]william.sl.tobing@gmail.com, [3]david.christiansitorus@gmail.com, [4]friskasianturi23@gmail.com
*Penulis Korespondensi

(Naskah masuk: 13 Desember 2020, diterima untuk diterbitkan: 21 Juli 2021)

Abstrak

Smartphone merupakan alat umum yang digunakan masyarakat dalam kehidupan sehari-hari. Sistem operasi yang paling banyak digunakan pada *smartphone* adalah Android. Aplikasi pada Android dapat diperoleh tidak hanya di Play Store. Namun, dapat juga diperoleh secara bebas di *website* yang berada di internet. Oleh karena itu aplikasi Android rentan terhadap pembajakan. *Software watermarking* merupakan metode umum yang biasanya digunakan untuk mengantisipasi pembajakan perangkat lunak dengan menyisipkan informasi pengenal ke dalam suatu program. Tujuan dari *software watermarking* adalah untuk membuktikan kepemilikan dari sebuah program. Salah satu teknik *watermarking* adalah *dynamic watermarking.* Teknik ini akan men-*generate watermark* ketika program dieksekusi. *Dynamic Graph Watermarking* (DGW) merupakan salah satu metode dalam *software watermarking*. Dalam penyisipan *watermark,* metode ini menggunakan struktur graf yang dibuat berdasarkan enumerasi graf. Salah satu algoritma dalam DGW adalah Colberg-Thomborson (CT) *algorithm.* Algoritma tersebut menggunakan *code* yang dapat membentuk *watermark* saat *runtime* program. Pemberian *watermark* terhadap sebuah aplikasi dilakukan dengan menggunakan CT *algorithm* dan enumerasi *Parent Pointer Graph (PPG).* Untuk menyisipkan *watermark* terhadap aplikasi Android, dibuat sebuah *library* Java dan sebuah simulator berbasis desktop untuk mengekstrak *watermark*. Dari hasil pengujian dapat disimpulkan bahwa PPG dapat digunakan sebagai enumerasi pada metode DGW dan memiliki tingkat ketahanan yang tinggi terhadap *distortive attack. N*amun, tidak pada *subtractive* dan *additive attack*. Dari penelitian juga diperoleh hasil bahwa pemberian *watermark* memberikan penambahan *size* pada aplikasi Android. Namun, tidak mempengaruhi peningkatan penggunaan *memory* dan *processor* aplikasi.

Kata kunci: Aplikasi Android, Pembajakan , DGW, CT *algorithm, PPG*

DYNAMIC SOFTWARE WATERMARKING WITH COLLBERG-THOMBORSON ALGORITHM AND PARENT POINTER GRAPH ON ANDROID APPLICATION

Abstract

Smartphones are common tools in people's daily life. The most common operating in smartphone is Android. Our Android application can be obtained not only in the Play Store, but also free websites on the internet. Therefore, Android applications are vulnerable to piracy. Software watermarking is a common method used to anticipate software piracy by inserting identifying information into a program. The purpose of software watermarking is to prove ownership of a program. One of the watermarking techniques is dynamic watermarking that generates watermarks when the program is executed. Dynamic Graph Watermarking (DGW) is one of the software watermarking methods. This method uses a graph structure which created based on graph enumeration in inserting the watermark. One of the DGW algorithm is Colberg-Thomborson (CT) which use code that can form a watermark at program run time. For watermarking an application, we use CT algorithm and Parent Pointer Graph (PPG) enumeration. To embed watermark to the Android application we create a Java library and a desktop-based simulator to extract watermark from Android application. Our result shows that PPG can be used as an enumeration and has robustness in defending against distortive attack but not to subtractive and additive attacks. We also get that watermark gives additional size to an Android application but it does not affect the increase in memory and processor usage.

Keywords: *Android applications, Watermark, Piracy, DGW, CT algorithm,PPG*

1. PENDAHULUAN

Saat ini *smartphone* sudah menjadi alat umum yang digunakan masyarakat dalam kehidupan sehari-hari (Zhang & Chen, 2014). Jumlah pengguna *smartphone* meningkat setiap tahun (Milijic, 2019). Di Indonesia sendiri pengguna *smartphone* juga ikut meningkat pesat. Menurut data lembaga riset Digital Marketing Emarketer, pada 2019 jumlah pengguna aktif *smartphone* di Indonesia mencapai sekitar 92 juta pengguna (Databook, 2019). Banyaknya jumlah *smartphone* diikuti dengan banyaknya aplikasi Android. Oleh karenanya selain Play Store yang telah disediakan Google, terdapat banyak sumber yang dapat dicari di internet untuk mendapatkan aplikasi pada Android. Oleh karena itu aplikasi Android rentan dibajak. Pembajakan aplikasi telah menimbulkan beberapa masalah seperti pelanggaran *copyright* dan masalah keaslian (Rosadi, 2009), yaitu penyalinan yang tidak berizin terhadap aplikasi target dan mengakuinya sebagai milik sendiri dan juga pemodifikasian aplikasi sehingga data-data yang terdapat dalam aplikasi telah berubah atau tidak sama lagi dengan versi *original*.

Pembajakan ini tentu merugikan pemilik aplikasi terutama saat pemilik tidak dapat membuktikan bahwa aplikasi tersebut adalah miliknya. Namun jika kepemilikan aplikasi dapat dibuktikan maka dapat ditemukan siapa pembajak aplikasi tersebut dan tidak menutup kemungkinan untuk menghentikan distribusi aplikasi bajakan tersebut. Oleh karena itu, dibutuhkan metode untuk melindungi *software* dari pembajakan (Sebastian, 2006).

Salah satu cara atau metode untuk melindungi suatu aplikasi dari pembajakan adalah dengan menyisipkan suatu *watermark* dalam aplikasi tersebut. Dengan keberadaan suatu *watermark* yang dapat diverifikasi maka pemilik aplikasi dapat mengetahui siapa yang sudah melakukan pembajakan terhadap aplikasinya (Sebastian, 2006). Teknik yang digunakan untuk menyisipkan *watermark* disebut *watermarking* (Chen, Jia, & Xu, 2017). Salah satu teknik *watermarking* adalah *Dynam*ic *watermarking.* Teknik ini menyisipkan *watermark* saat program *runtime.* Salah satu teknik *dynamic watermarking* adalah *Dynamic Graph Based Software Watermarking* dimana teknik ini menyisipkan *watermark* ke dalam topologi sebuah graf. Salah satu algoritma dalam *Dynamic Graph Based Software watermarking* adalah Collberg-Thomborson (CT) *Algorithm*. CT *Algorithm* terdiri dari 4 langkah utama yaitu *annotation, tracing*, *embedding* dan *extraction*. Dalam penelitian ini penulis melakukan proses *annotation, tracing* dengan menentukan posisi yang tepat untuk meletakkan *watermark* dan *embedding* dengan memasukkan *watermark* dalam aplikasi Android dan *extraction* dengan menggunakan sebuah simulator.

Untuk menyisipkan *watermark* ke dalam graf dilakukan dengan enumerasi. Dalam penelitian ini akan digunakan *Dynamic Graph Based Software Watermarking* dengan CT *Algorithm* dan enumerasi *Parent Pointer Graph (PPG).* PPG merupakan teknik enumerasi di mana setiap simpul memiliki *pointer* yang mengarah ke induknya (He, 2002). Enumerasi ini dipilih karena lebih sederhana daripada enumerasi lainnya seperti *permutation encoding* dan Radix *Encoding*(Fathiya & Munir, 2013). Berdasarkan penelitian Shofi Nur Fathiya dan Rinaldi Munir pembuatan graf dibuat dari *string watermark*. Dimana setiap karakter *watermark* ini akan diubah ke dalam susunan angka menggunakan kode ASCII. Namun, dalam penelitian ini dilakukan perubahan pada aturan-aturan pembuatan susunan angka karena pada beberapa kasus aturan pengubahan angka pada penelitian sebelumnya menghasilkan susunan angka yang salah pada saat pengekstrakan. Untuk membentuk angka-angka pembangun graf *watermark* penulis membuat sebuah *library* Java yang berisikan kelas-kelas Android untuk men-*generate* sebuah *file* .txt. *File* .txt akan ter-*generate* setelah aplikasi ter-*install*. *Library* ini akan digunakan saat *development* aplikasi yang artinya *watermark* akan disisipkan ke dalam *code base* aplikasi. Sebuah *watermark* harus kuat terhadap *attacking*. *Attacking* merupakan upaya merusak *watermark.* Dalam penelitian ini, penulis menguji ketahanan CT *Algorithm* terhadap serangan *distortive*, *additive* dan *subtractive* dan juga meneliti pengaruh pemberian *watermark* terhadap peningkatan *size* aplikasi Android, penggunaan *memory* dan *processor.*

2. METODE PENELITIAN

Bab ini terdiri dari subbab *dataset*, pembuatan graf *watermark* dengan PPG, penyisipan *watermark*, ekstraksi *watermark* dan pengujian *watermark.*

2.1. *Dataset*

Pada penelitian ini *watermark* yang akan disisipkan adalah sebuah *string*. Tabel 1 menunjukkan *dataset watermark* yang digunakan

Tabel 1. Dataset Watermark

No	Watermark
1	a
2	rop
3	TADe3
4	*(?12

Karakter-karakter penyusun *string watermark* berasal dari ASCII *printable characters* seperti Gambar 1 berikut

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Gambar 1. Table ASCII
(The ASCII Code, 2007)

Dataset *watermark* merupakan sekumpulan *string* yang dirancang penulis sendiri. *Dataset* tersebut telah mewakili huruf kecil, huruf besar, angka dan juga simbol. Tabel 2 berikut *dataset* aplikasi Android yang digunakan

Tabel 2. Dataset Android

No	Nama Aplikasi
1	E-CommerceApp(Prathapan, 2018)
2	FilmKu(Manurung, 2019)
3	GroceryStore(Martinez, 2017)
4	HomeMarket(InnovaTechno, 2019)
5	LaptopArena(LumbanTobing B.2020)
6	Memommond(Lumbantobing, W. 2020a)
7	PhoneArena(Lumbantobing, W. 2020b)

Dataset *project* aplikasi Android yang digunakan berasal dari GitHub.

2.2. Pembuatan Graf *Watermark* Dengan PPG

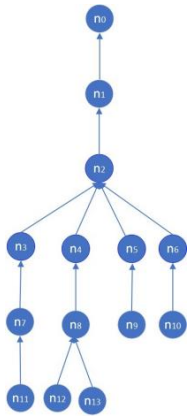
Teks *watermark* diubah menjadi sekumpulan angka yang akan direpresentasikan menjadi graf dengan menggunakan kode ASCII. Adapun langkah-langkahnya dilakukan sebagai berikut (Fathiya & Munir 2013).

1. Mengubah setiap karakter ke dalam angka menggunakan karakter ASCII.
2. Menggabungkan hasil pengubahan setiap karakter menjadi deretan angka.
3. Mengubah deretan angka menjadi angka-angka pembentuk graf *watermark* dimana setiap angka mewakili jumlah *child* yang dimilikinya.

Contohnya sebuah *watermark* "rop"

1. 'r' memiliki nilai ASCII 114, 'o' bernilai 111 dan 'p' bernilai 112.
2. Deretan angka yang dihasilkan adalah 114,111 dan 112. Sehingga string angka yang dihasilkan adalah 114111112.
3. Untuk membuat susunan angka membentuk graf, maka setiap digit angka akan menjadi *child* dari angka sebelumnya. *Node* awal akan dimulai dari *node* n0. Contohnya 114111112, angka 1 pada index [0] akan menjadi *child* dari n0 sehingga menghasilkan n1, angka 1 pada index [1] akan menjadi *child* dari n1 menghasilkan n2, angka 4 pada index [2] menjadi *child* dari n2 menghasilkan n3, n4, n5, n6, dst

Graf yang dihasilkan untuk *watermark* ini ditunjukkan oleh Gambar 2.



Gambar 2. Graf Watermark "rop"

Pada penelitian ini terdapat sedikit perubahan dari penelitian sebelumnya. Pengubahan ini bertujuan agar karakter-karakter yang ingin dijadikan *watermark* dapat dibuat ke dalam graf dan nantinya dapat diekstrak sesuai dengan karakter awal. Dalam penelitian sebelumnya, terdapat kasus yang menyebabkan *watermark* yang diekstrak tidak sesuai dengan keadaan semula. Sebagai contoh *watermark* yang digunakan adalah "sds". ASCII dari karakter 's' adalah 115 dan 'd' adalah 100 maka susunan angkanya 115100115. Angka 0 tidak dapat membentuk *node* graf sehingga susunan deretan angkanya adalah 1151115. Lalu saat pengekstrakan susunan angkanya ditambah 2 angka 0 sehingga panjangnya menjadi kelipatan 3. Susunan angkanya menjadi 115111500. Setelah diubah kembali ke karakter menggunakan ASCII menjadi 115 = 's', 111 = 'o' dan 500-300 adalah 200 dan menghasilkan karakter lain yang bukan merupakan huruf atau angka yang sering digunakan sebagaimana ada dalam printable ASCII. Hal ini membuat *watermark* yang dihasilkan salah. Dalam pengubahan sebuah karakter menjadi sekumpulan angka menggunakan ASCII ada beberapa aturan yang penulis buat agar teks *watermark* dapat disisipkan dan diekstrak dengan memiliki teks yang sama, yaitu:

a. Penyisipan
   1. Jika nilai ASCII habis dibagi 10 maka akan ditambah 301.
   2. Jika nilai ASCII lebih dari 100 dan tidak habis dibagi 10 maka akan ditambah 10.
   3. Angka 100 akan dikodekan menjadi 138.
   4. Jika nilai ASCII terdiri dari dua angka dan di modulus sepuluh hasilnya tidak sama dengan 0 maka ditambah 200

b. Pengekstrakan

Pada pengekstrakan, graf diubah menjadi susunan angka. Kemudian, akan diambil *substring* sepanjang tiga karakter dari kumpulan angka tersebut. Langkah selanjutnya, *substring* tersebut diubah menjadi integer lalu angka tersebut dikembalikan ke dalam karakter melalui ASCII. Pengubahan angka menjadi nilai ASCII kembali mengikuti salah satu aturan berikut. Setiap tiga digit angka dikonversi ke karakter dengan menggunakan ASCII masing-masing**.** Dengan aturan sebagai berikut:

1. Jika angka lebih dari 300, maka akan dikurang 301.
2. Jika angka lebih dari 200 dan lebih kecil dari 300 maka akan dikurang 200.
3. Angka angka yang lebih dari 100 dan lebih kecil dari 200 akan dikurang 10.
4. Angka 100 akan menjadi 138.

2.3. Penyisipan *Watermark*

Pada penyisipan *watermark* terdiri dari 2 bagian, yaitu: proses pembuatan *library* dan penyisipan *watermark* dengan menggunakan *library* yang dibuat pada aplikasi Android.

2.3.1 *Library* Penyisipan *Watermark*

Pada penelitian sebelumnya, penyisipan *watermark* dilakukan dengan mendesain graf dari susunan angka yang ada. Kemudian, memasukkan semua angka pada graf yang telah di desain sebelumnya ke dalam program. Sementara pada penelitian ini, pengguna tidak perlu memasukkan semua angka pada graf tersebut tapi, hanya perlu memasukkan *string watermark.* Angka-angka pembentuk graf *watermark* akan dibentuk oleh program sendiri. *Library* untuk menyisipkan *watermark* terdiri dari 3 kelas, yaitu: Watermark, Hasil dan Encoder.

a. Kelas Watermark bertanggung jawab sebagai *watermark* sendiri

```
public class Watermark {
    public String nama_watermark;
    public Watermark(String watermark){
        nama_watermark = watermark;
    }
}
```

b. Kelas Encoder berfungsi untuk mengubah *watermark* menjadi sekumpulan angka dengan menggunakan ASCII dan pengecekan terhadap syarat-syarat yang dibuat penulis

```
public class Encoder {
public static String encodeString(String watermark
){
        String watermarkAscii="";
        for (char c : watermark.toCharArray()){
            watermarkAscii
+=Integer.toString(cekKodeEncode((int)c));
        }
        return watermarkAscii;
    }
private static int cekKodeEncode(int ascii){
        if(encodeAsciiSeratus(ascii)){
            ascii = 138;
        }else if(encodeAsciiLebihSeratus(ascii)){
            ascii+=10;
        }else if(encodeAsciiDuaAngka(ascii)){
            ascii+=200;
        }else
if(encodeAsciiHabisModSepuluh(ascii)){
            ascii+=301;
        }
        return ascii;
    }
private static boolean encodeAsciiSeratus(int
ascii){
        if(ascii == 100){
            return true;
        }
        return false;
    }
    private static boolean encodeAsciiSeratus(int
ascii){
        if(ascii == 100){
            return true;
        }
        return false;
    }
```

```
 private static boolean encodeAsciiLebihSeratus(int
ascii){
        if(ascii > 100 && ascii <200 &&    ascii
%10 !=0){
            return true;
        }
        return false;
    }

private static boolean encodeAsciiDuaAngka(int
ascii){
        if(ascii < 100 &&  ascii % 10 != 0){
            return true;
        }
        return false;
    }
    private static Boolean
encodeAsciiHabisModSepuluh(int ascii){
        if(ascii % 10 == 0){
            return true;
        }
        return false;
    }
```

c. Kelas Hasil merupakan kelas yang akan men-*generate file* .txt yang akan berisikan angka-angka pembangun graf *watermark*

```
public class Hasil {
    public static ArrayList <Pair <Integer,Integer>
> data = new ArrayList <Pair <Integer,Integer> >
();

    public static int counterhuruf = 0;
    static String path =
Environment.getExternalStorageDirectory().getAbsolu
tePath() + "/Watermark";
    static FileOutputStream  outStream = null;
    static OutputStreamWriter  outStreamWriter =
null;

    public static void createWriter() throws
IOException {
        File dir = new File(path);
        dir.mkdirs();
        File file = null;
        file = new File(new File(path),
"/WatermarkTAII.txt");
   file.createNewFile();
        outStream = new FileOutputStream(file,
true);
        outStreamWriter = new
OutputStreamWriter(outStream);
    }

    public static void buildWatermark( String
watermark) throws IOException {
        createWriter();
        watermark =
Encoder.encodeString(watermark);
        int ctr=2;
        for(char c: watermark.toCharArray()){
            counterhuruf++;
            int character_angka =
Character.getNumericValue(c);
            for(int i = counterhuruf; i <
counterhuruf + character_angka ;i++){
                outStreamWriter.append(counterhuruf
+" "+ ctr+"\n");
                ctr++;
            }
        }

         outStreamWriter.flush()
    }

}
```

2.3.2 Penyisipan *Watermark* Pada Aplikasi Android

Dalam aplikasi Android akan dilakukan pemanggilan fungsi watermark() yang akan menyisipkan *watermark. Watermark* yang disisipkan disimpan oleh *variable watermark yang* diimplementasikan seperti berikut.

```
private void watermark() throws IOException {
    String watermark = "friska";
    Watermark kata= new Watermark(watermark);
    Hasil.buildWatermark(kata.nama_watermark);
}
```

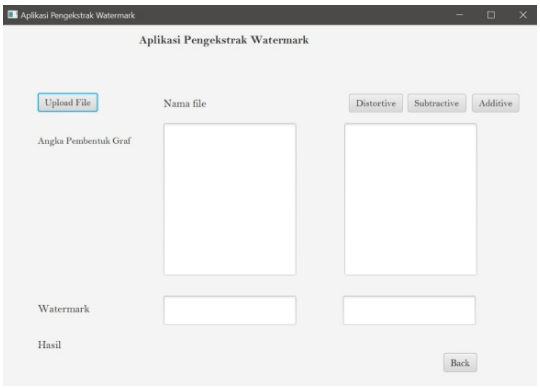
Jika aplikasi di-*install*, maka pada aplikasi akan ter-*generate file* .txt yang berisikan angka-angka pembentuk graf *watermark* seperti Gambar 3 berikut.

Gambar 3. txt Graf Builder Watermark

Dengan adanya *file* .txt pada penyimpanan lokal menandakan bahwa *watermark* telah tersisip pada aplikasi.

2.4. Ekstraksi *Watermark*

Pengekstrakan dilakukan oleh sebuah simulator. Gambar 4 berikut merupakan tampilan simulator yang telah dibuat.



Gambar 4. Simulator Watermark

Pada simulator terdapat 3 *button* yang berfungsi untuk melakukan serangan pada *watermark. Button* Distortive berfungsi untuk melakukan serangan *distortive*, *button* Substractive berfungsi untuk melakukan serangan *subtractive* dan *button* Additive untuk melakukan serangan *additive. Button* Distortive untuk melakukan serangan *distortive*, dilakukan dengan mengacak angka-angka pembentuk *watermark*. Serangan *subtractive* dilakukan dengan menghapus simpul terakhir angka dan serangan *additive* dilakukan dengan memberikan tambahan 1 simpul pada susunan angka-angka pembentuk graf. Berikut merupakan implementasi serangan yang dilakukan oleh penulis

```
public class Attack {
    public static List<String>
distortive(List<String> listEdge)
    {
        Collections.shuffle(listEdge);
      return listEdge;
    }
    public static List<String>
additive(List<String> listEdge, String addition)
    {
        listEdge.add(addition);
      return listEdge;
    }
    public static List<String>
subtractive(List<String> listEdge)
    {
        for (String string : listEdge) {
            listEdge.remove(string);
            break;
        }
      return listEdge;
    }
}
```

Ekstraksi *watermark* dilakukan dengan menerima *file* .txt yang dihasilkan dari instalasi aplikasi yang telah tersisip *watermark*. Angka-angka dalam *file* akan diubah menjadi graf. Dari graf yang terbentuk akan diperoleh susunan angka pembentuk *watermark* awal. Susunan angka akan di-*decode* menggunakan ASCII. Proses *decode* merupakan kebalikan dari *encode*. Simulator dilengkapi dengam label Hasil. Setelah proses ekstraksi selesai label ini akan menampilkan kata "sama" jika *watermark* yang diekstrak sama dengan *watermark* yang telah diberi serangan dan "tidak sama" jika tidak.

2.5. Pengujian *Watermark*

Bab ini terdiri dari pengujian ketahanan *watermark*, peningkatan *size* aplikasi serta penggunaan *memory* dan *processor* pada aplikasi yang telah kesisipan *watermark*.

2.5.1. Pengujian Ketahanan *Watermark*

Pengujian ketahanan *watermark* dilakukan dengan menggunakan simulator. Pada simulator terdapat 3 *button* untuk melakukan 3 serangan, yaitu: *distortive, additive* dan *subtractive.* Tabel 3 menampilkan hasil ketahanan *watermark* terhadap 3 buah serangan

Tabel 3. Hasil Pengujian Ketahanan

No	Aplikasi	Serangan		
		Subtractive	Distortive	Additive
1	E-Comm erceApp	X	O	X
2	FilmKu	X	O	X
3	GroceryStore	X	O	X
4	HomeMarket	X	O	X
5	LaptopArena	X	O	X
6	Memommond	X	O	X
7	PhoneArena	X	O	X

Keterangan:
O = watermark dapat diambil secara utuh
X = watermark tidak dapat diambil secara utuh

Pada pengujian, *watermark* tahan terhadap serangan *distortive.* Namun, pada serangan yang lainnya *watermark* tidak bisa diekstrak secara utuh dan ada *watermark* yang tidak dapat diekstrak sama sekali.

2.6. Eksperimen Peningkatan *Size*

Analisis peningkatan *size* aplikasi(.apk) dianalisis dengan menggunakan fitur Android studio Analyze .apk. Fitur ini akan menampilkan *Diff Size* (perbedaan *size* dua buah .apk) dari apk yang belum dan sudah memiliki *watermark*. Pada Tabel 4 terlihat bahwa penyisipan *watermark* memberikan rata-rata peningkatan *size* sebesar 7.413 KB

Tabel 4. Rata-rata Diff Size Apk Yang Telah Diberi Watermark

No	Nama Aplikasi	Avarage Diff Size(KB)
1	E-Com merceApp	6
2	FilmKu	8
3	GroceryStore	7,75
4	HomeMarket	7
5	LaptopArena	7,75
6	Memommond	6,75
7	PhoneArena	6,75
	Avarage	7,413

2.7. Eksperimen Penggunaan *Memory* dan *Processor*

Peningkatan penggunaan *memory* dan *processor* menggunakan fitur Profiler pada Android Studio. Table 5 berikut menampilkan data penggunaan *memory* tertinggi dan *processor* dari aplikasi yang tersisip *watermark.*

Tabel 5. Penggunaan *Memory* Dan *Processor*

No	Nama Apk	Memory Tertinggi(MB)	Thread pada Processor 1 Menit
1	E-Comm erceApp	0	0
2	FilmKu	0	0
3	GroceryStore	0	0
4	HomeMarket	0	0
5	LaptopArena	0	0
6	Memommond	0	1
7	PhoneArena	0	0

3. TINJAUAN PUSTAKA

Pada tinjauan pustaka dijelaskan mengenai rangkuman teori-teori yang mendukung penelitian.

3.1 Android

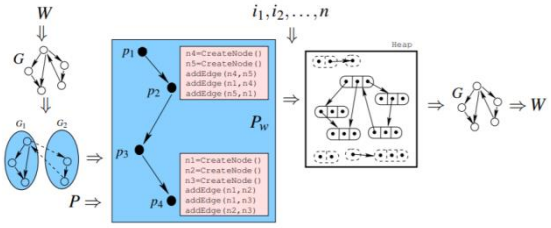
Android adalah sistem operasi open source, Google merilis kodenya di bawah Lisensi *Apache.* Laporan terakhir dari Google yaitu pada *event* Google I/O 2019, pengguna Android telah mencapai 2,5 miliar pengguna (The Verge, 2019). Pengguna yang begitu banyak menjadikan Android sistem operasi terpopuler dan paling banyak penggunanya di dunia.

3.2 *Dynamic Graph Based Software Watermarking*

Dynamic software watermarking merupakan salah satu tindakan pencegahan utama terhadap pelanggaran lisensi perangkat lunak(Ma et al. 2019). Salah satu teknik dynamic *software watermarking* adalah *Dynamic Graph based Software watermarking.* Teknik ini menyisipkan *watermark* ketika *runtime* program sehingga *watermark* tidak mudah terdeteksi. Teknik ini menggunakan struktur graf yang dibuat berdasarkan enumerasi graf (Fathiya & Munir, 2013).

3.3 CT *Algorithm*

CT *Algorithm* merupakan algoritma dari *Dynamic Graph Based Watermarking*. Gambar 5 berikut merupakan konsep CT *algorithm*

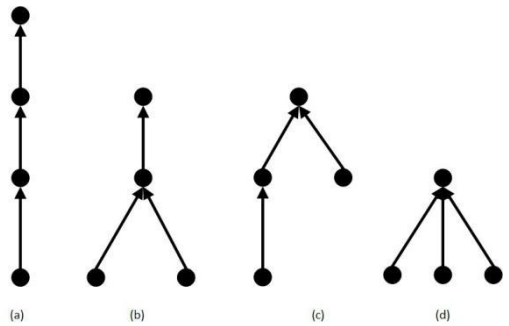


Gambar 5. Overview CT Algorithm

Ada 4 langkah dalam algoritma ini yaitu *annotation*, *tracing*, *embedding*, dan *extraction* (Collberg, Thomborson, and Townsend 2004). *Annotation* merupakan penentuan titik penyisipan *watermark*. *Tracing* merupakan penelusuran di mana titik-titik tersebut berada. *Embedding* merupakan penyisipan *watermark* ke dalam aplikasi dan *extraction* atau pengekstrakan merupakan pengambilan *watermark* dari aplikasi yang sudah disisipkan *watermark.*

3.4 *Parent Pointer Graph* (PPG)

Pada PPG setiap simpul memiliki satu *pointer* ke arah induknya (He, 2002). Gambar 6 menunjukkan graf PPG yang dapat dibangun dengan menggunakan 4 simpul.



Gambar 6. Graf 4 Simpul

3.5 Serangan (*Attacks*)

Terdapat beberapa jenis serangan pada metode *Dynamic Graph Watermarking* yang dapat diterapkan untuk pengujian ketahanan *watermark* (He, 2002). Serangan dapat dikategorikan menjadi beberapa bagian sebagai berikut.

a. *Subtractive attack*
   Pada *subtractive attack*, penyerang menghilangkan *watermark* dari aplikasi yang diberi *watermark*.
b. *Distortive attack*
   Pada *distortive attack*, penyerang mengacak *watermark* sehingga *watermark* tidak dapat dikenali lagi.
c. *Additive attack*
   Pada serangan adiktif, penyerang menambahkan *watermark* baru kedalam program yang sudah diberi *watermark.*

4. HASIL DAN PEMBAHASAN

Pada eksperimen menunjukkan bahwa *watermark* telah berhasil tersisip pada aplikasi Android dengan menggunakan DGW, CT *Algorithm* dan enumerasi PPG. Pembentukan angka-angka *watermark* dengan menggunakan PPG dapat menghasilkan angka-angka yang dapat disisipkan dan saat diekstrak menghasilkan *watermark* yang sama dengan *watermark* ketika disisipkan. Dalam eksperimen ketahanan diperoleh hasil bahwa *watermark* yang dibuat tahan terhadap serangan *distortive*. Namun, tidak tahan terhadap serangan *additive* dan *subtractive*. Untuk *decode* sebuah *watermark* digunakan angka-angka pembentuk graf *watermark* yang disimpan dalam *Adjacency List*. Pada serangan *distortive*, pengacakan hanya menyebabkan posisi angka setiap baris berbeda. Namun, komposisi angka-angka pembentuk graf adalah sama sehingga *size* dari *list* penampung sebuah simpul dan childnya di *Adjacency List* yang digunakan masih sama sedangkan pada serangan *distortive* dan *additive* komposisi angka pembentuk graf sudah berbeda sehingga *size* dari *list* penampung sebuah simpul dan *child* simpul yang digunakan juga menjadi berbeda. *Watermark* yang tidak dapat diekstrak disebabkan oleh angka-angka yang terbentuk tidak membentuk susunan angka kelipatan 3 sehingga tidak dapat diubah ke dalam karakter menggunakan ASCII. Dari rata-rata penggunaan *memory* dan *processor* setiap aplikasi pemberian *watermark* tidak memengaruhi penggunaan *memory* dan *thread* pada *processor*. Pada peningkatan *size* rata-rata peningkatan *size* yang terjadi adalah sebesar 7.413 atau 0,0069MB

5. Kesimpulan

Pada penelitian ini, *Dynamic Software Watermarking* dapat diimplementasikan dalam aplikasi Android dengan CT *Algorithm* dan PPG. Kemudian, diperoleh hasil *watermark* yang sama saat *watermark* tersisip dan diekstrak. Eksperimen juga menunjukkan bahwa *watermark* dengan CT *Algorithm* dan enumerasi PPG tahan terhadap serangan *distortive* tapi tidak pada *additive* dan *subtractive.* Pada Android sendiri penyisipan *watermark* memberi pengaruh terhadap *size* aplikasi. Rata-rata perbedaan *size* yang diperoleh setelah menyisipkan *watermark* adalah 7,143KB atau 0,006976MB. Untuk penggunaan *memory* dan *processor* setelah pemberian *watermark*, aplikasi Androidtidak menunjukkan perubahan yang dapat mempengaruhi penggunaan *memory* dan *processor* pada perangkat terpasang. Dan pada modifikasi aturan angka yang dilakukan pada penelitian Fathiya dan Munir, penulis berhasil membentuk susunan angka yang berbeda dengan *string* yang sama sehingga mencakup seluruh nilai ASCII yang ada di ASCII *printable characters*, mulai dari dari 32 hingga 126

DAFTAR PUSTAKA

THE ASCII CODE. 2007. ASCII table, ascii codes. Tersedia di <https://theasciicode.com.ar/>

CHEN, Z., JIA, C. AND XU, D. 2017. Hidden Path: Dynamic Software Watermarking Based on Control Flow Obfuscation.

COLLBERG, C., THOMBORSON, C. AND TOWNSEND, G. M. 2004. Dynamic Graph-Based Software Watermarking.pp. 5–12.

DATABOOKS. 2019. Pengguna Smartphone Di Indonesia 2016-2019. Tersedia di <https://databoks.katadata.co.id/datapublish/2016/08/08/pengguna-smartphone-di-indonesia-2016-2019>

FATHIYA, S, N., AND MUNIR, R. 2013. Penggunaan Parent Pointer Graph Untuk Software Watermarking Berbasis Graf Dinamis.pp. 1–5.

ROSADI, F, A,. 2009. Studi Tentang Software Watermarking.pp. 1–7.

HE, Y. 2002. Tamperproofing a Software Watermark by Encoding Constants.

INNOVATECHNO. 2019. Home Market. https://github.com/InnovaTechno/HomeMarket.

LUMBANTOBING, B. 2020. Laptop Arena. https://github.com/bryantobing12/Laptop-Arena.

LUMBANTOBING, W. 2020a. Memommond. https://github.com/williamtobing/Memommond.

LUMBANTOBING, W 2020b.Phone Arena. https://github.com/williamtobing/Application-Embedded-Watermark.

MA, H, JIA,C, LI, S, ZHENG, W AND WU, D. 2019. Xmark: Dynamic Software Watermarking Using Collatz Conjecture.

MANURUNG, N. 2019. Film Ku. https://github.com/nick2905/Submission_1.

MARTINEZ, A, G,. 2017. Grocery Store. https://github.com/czyrux/GroceryStore.

MILIJIC, M. 2019. 29+ Smartphone Usage Statistics: Around the World in 2020. <https://leftronic.com/smartphone-usage-statistics/>

PRATHAPAN, P. 2018. E-CommerceAndroidAppExample. https://github.com/preethzcodez/E-CommerceAndroidAppExample.

SEBASTIAN, A. 2006. Penggunaan Watermarking Pada Penyebaran Software Untuk Perlindungan Hak Cipta.pp. 1–7.

THE VERGE. 2019. There Are Now 2.5 Billion Active Android Devices. Tersedia di <https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote>

ZHANG, YINGJUN, AND CHEN, K,. 2014. AppMark : A Picture-Based Watermark for Android Apps.