Studi dan Perbandingan Algoritma *Software Watermarking* Berbasis Graf dan *Opaque Predicate*

Julian Sukmana Putra [1)]

1) Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, Bandung 40132, email: if14143@students.if.itb.ac.id

Abstract **–** *Software watermarking* adalah salah satu metode yang diharapkan dapat mengurangi tingkat pembajakan perangkat lunak. Hingga saat ini, beragam algoritma *software watermarking* telah diajukan dan diimplementasikan. Secara umum, terdapat dua kategori algoritma *software watermarking*, yaitu statis dan dinamis. Algoritma *software watermarking* dinamis memiliki kelebihan dibandingkan algoritma statis pada ketahanannya terhadap serangan distortif. Makalah ini menyajikan hasil studi dan perbandingan dua algoritma *software watermarking*, yaitu algoritma yang berbasis graf dan algoritma yang menggunakan *opaqure predicate*. Perbandingan dilakukan dengan mengacu pada kerangka kerja yang diusulkan dalam [1] dan menggunakan perangkat lunak SandMark.

Kata Kunci*: software watermarking, algoritma, graf, opaque predicate, SandMark*

1. PENDAHULUAN

Dalam industri perangkat lunak, pembajakan dan pelanggaran hak cipta pada perangkat lunak merupakan permasalahan yang serius. Seiring dengan perkembangan jaringan Internet, pendistribusian perangkat lunak menjadi lebih mudah, sehingga tingkat kejahatan tersebut terus bertambah. Selain itu, kini perangkat lunak dapat secara legal didistribusikan dalam format yang tidak terikat pada *platform* seperti bytecode Java dan Microsoft's Intermediate Language (MSIL). Format tersebut sangat menyerupai kode sumbernya, sehingga dapat dengan mudah dimanipulasi dan direkayasa-balik oleh para pembajak.

Software watermarking adalah salah satu metode yang diharapkan dapat mengurangi tingkat pembajakan perangkat lunak. Hal ini dilakukan dengan menyisipkan informasi rahasia dalam perangkat lunak tersebut, misalnya identitas pembuat perangkat lunak atau informasi mengenai pengguna perangkat lunak yang sah. Dengan metode ini, kemungkinan untuk membuktikan tindakan pembajakan dan pelanggaran hak cipta pada perangkat lunak menjadi lebih besar.

Watermarking merupakan aplikasi dari steganografi [5] untuk melindungi data dan informasi yang disajikan dalam format digital, baik berupa teks, citra, audio, maupun video. Penerapan *watermarking* pada perangkat lunak serupa dengan *watermarking* dalam media digital. Namun, dalam *software watermarking*, penyisipan informasi tersebut harus dapat menjaga fungsionalitas perangkat lunak yang disisipi dan memiliki tingkat keamanan yang tinggi, sehingga dapat bertahan dari berbagai serangan.

Sebuah sistem s*oftware watermarking* dapat didefinisikan sebagai berikut [3]:

Definisi 1 (sistem *software watermarking*) . Diberikan sebuah program *p*, sebuah *watermark w*, dan sebuah kunci *k*, sebuah sistem *software watermarking* terdiri dari dua fungsi, yaitu

$$\text{embed}(p,w,k) \rightarrow p'$$
$$\text{recognize}(p',k) \rightarrow w$$

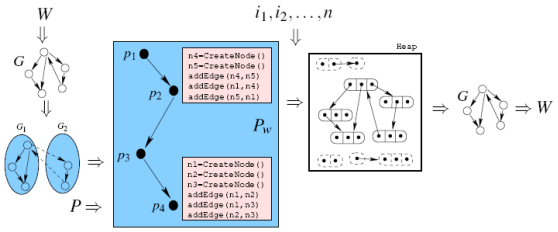
Secara umum, terdapat dua kategori algoritma *software watermarking*, yaitu statis dan dinamis. Algoritma statis hanya menggunakan fitur aplikasi yang tersedia pada saat kompilasi, seperti rangkaian instruksi atau *constant pool table* pada aplikasi Java, sedangkan algoritma dinamis, penyisipan dan pengenalan *watermark* bersandar pada informasi yang dikumpulkan selama pengeksekusian aplikasi tersebut. Berbagai studi mengenai kedua kategori algoritma di atas belum dapat menyimpulkan mana algoritma yang lebih tinggi tingkat keamanannya untuk *software watermarking* [3]. Namun, secara umum telah diketahui bahwa algoritma statis rentan terhadap serangan distortif pada kode sumber seperti *code obfuscation* [1]. Algoritma dinamis relatif lebih tahan terhadap jenis serangan tersebut.

Makalah ini menyajikan hasil studi dan perbandingan dua algoritma *software watermarking* dinamis, yaitu algoritma CT (Collberg-Thomborson), sebuah algoritma *software watermarking* yang berbasis graf, dan *Dynamic* Arboit, algoritma *software watermarking* yang menggunakan *opaque predicate*. Perbandingan kedua algoritma tersebut didasarkan pada kerangka kerja yang diusulkan dalam [1].

2. ALGORITMA *SOFTWARE WATERMARKING* BERBASIS GRAF

Algoritma CT adalah salah satu algoritma *software watermarking* dinamis yang berbasis graf [2]. Ide dasar dari algoritma ini ialah dengan menyisipkan

kode yang dapat mengonstruksi *watermark* pada saat *runtime*. Hal ini berbeda dengan algoritma statis yang menyisipkan secara langsung *watermark* pada kode sumber program. Algoritma ini mengasumsikan sebuah kunci rahasia *k* yang dibutuhkan untuk mengekstraksi *watermark*. *K* adalah serangkaian masukan I_0, I_1, … ke dalam program. Proses penyisipan dan pengekstraksian *watermark* dapat dilihat pada gambar di bawah ini



Gambar 1: Proses penyisipan dan pengekstraksian *watermark* pada algoritma CT

Sebuah angka *watermark W* disisipkan ke dalam topologi dari sebuah graf *G*. Graf tersebut dibagi menjadi beberapa komponen G_1, G_2, …. Setiap G_i dikonversi menjadi bytecode Java C_i yang membangun komponen graf tersebut. Kemudian, setiap C_i disisipkan ke dalam aplikasi di sepanjang jalur pengeksekusian yang diambil berdasarkan masukan khusus I_0, I_1, ….

Selama proses ekstraksi, aplikasi yang telah disisipi *watermark* berjalan dengan I_0, I_1, … sebagai masukan. Graf dari *watermark* tersebut dibangun dalam *heap*, kemudian graf tersebut diekstraksi, sehingga angka *watermark* yang disisipkan ditemukan.

Berikut ini adalah contoh program sederhana yang mengalami perubahan setelah disisipi *watermark*.

```
public class Simple {
   static void P(String i) {
      System.out.println("Hello, " + i);
   }
   public static void main(String args[]){
      P(args[0]);
   }
}
```

↓

```
public class Simple_W {
   static void P(String i, Watermark  n2) {
      if (i.equals("World")) {
         Watermark n1 = new Watermark();
         Watermark n3 = new Watermark();
         n4.edge1 = n1;
         n1.edge1 = n2;
         Watermark n3 = (n2 != null) ?
            n2.edge1 : new Watermark();
         n3.edge1 = n1;
      }
      System.out.println("Hello, " + i);
   }

   public static void main(String args[]) {
      Watermark n3 = new Watermark();
      Watermark n2 = new Watermark();
      n2.edge1 = n3;
      n2.edge2 = n3;
      P(args[0], n2);
   }
}

class Watermark extends java.lang.Object {
   public Watermar edge1, edge2;
}
```

Gambar 2: Contoh penyisipan *watermark* pada program Java

Algoritma CT yang digunakan untuk perbandingan dalam makalah ini adalah hasil implementasi yang digunakan dalam perangkat lunak SandMark 3.40. Proses penyisipan dan pengekstraksian *watermark* pada algoritma CT berjalan dalam beberapa langkah, yaitu:

Anotasi (*Annotation*)
Sebelum *watermark* dapat disisipkan, pengguna harus menambahkan titik anotasi ke dalam aplikasi yang akan diberi *watermark*, yaitu berupa pemanggilan fungsi tertentu seperti di bawah ini

```
mark();
String s = ...;
mark(s);
long l = ...;
mark(l);
```

Gambar 3: Contoh penambahan anotasi *watermark* ke dalam program Java

Pemanggilan `mark();` tidak akan menghasilkan aksi apapun. Pemanggilan tersebut hanya memberikan indikasi bagi pemberi *watermark* lokasi dalam kode di mana bagian dari kode pembangun *watermark* dapat disisipkan. Argumen dalam pemanggilan `mark();` dapat berupa ekspresi string maupun integer apapun yang bergantung pada masukan dari pengguna.

Penelusuran (*Tracing*)
Ketika aplikasi telah diberi anotasi, pengguna melakukan proses penelusuran (*tracing run*) program tersebut. Aplikasi berjalan dengan serangkaian masukan khusus *I*. Selama aplikasi berjalan, satu atau lebih titik anotasi akan terkena. Beberapa dari titik-titik tersebut akan menjadi lokasi di mana kode pembangun *watermark* kemudian disisipkan.

Penyisipan (Embedding)
Pada waktu proses penyisipan, pengguna memasukkan sebuah *watermark* yang berupa string maupun integer. Sebuah string akan dikonversi menjadi integer. Dari angka ini, kemudian dibuat graf sedemikian, sehingga topologi dari graf menyimpan angka tersebut. Graf ini dikonversi menjadi bytecode Java yang membangun-nya. Pemanggilan `mark();` yang relevan digantikan oleh kode ini.

Pengekstraksian (Extraction)
Pada saat proses pengekstraksian, aplikasi dijalankan lagi dengan masukan rahasia yang telah ditentukan.

Lokasi-lokasi `mark();` yang sama akan terkena selama penelusuran berlangsung. Namun, kini lokasi-lokasi tersebut akan berisi kode-kode untuk membangun graf *watermark*. Ketika bagian terakhir dari masukan telah diberikan, *heap* diperiksa untuk menemukan graf yang potensial sebagai graf *watermark*. Graf tersebut diuraikan dan angka *watermark* yang dihasilkan dilaporkan kepada pengguna.

3. ALGORITMA *SOFTWARE WATERMARKING* DENGAN *OPAQUE PREDICATE*

Penggunaan *opaque predicate* untuk *software watermarking* pertama kali diajukan oleh Monden [3]. Secara informal, *opaque predicate* disisipkan ke dalam kode sumber program untuk mempersulit lawan saat menganalisis alur-kontrol dari aplikasi. Hal ini membuatnya sulit untuk mengidentifikasi bagian tertentu dari program yang tidak berguna bagi sistem secara fungsional, namun bagian tersebut biasa digunakan untuk menyisipkan *watermark*.

Definisi 2 (*opaque predicate*). Sebuah predikat *P* bersifat *opaque* pada suatu titik *p* dalam program, jika pada titik *p* hasil dari *P* diketahui pada saat penyisipan.

Definisi 3 (*opaque method*). Sebuah *method* boolean *M* bersifat *opaque* pada sebuah titik invokasi *p*, jika pada titik *p* jika pada titik *p* nilai kembali dari *M* diketahui pada saat penyisipan.

Tantangan utama dalam penggunaan *opaque predicate* atau *opaque method* ialah bagaimana membuatnya tahan terhadap berbagai bentuk analisis yang dilakukan oleh lawan untuk menemukannya. Jika lawan dapat dengan mudah menguraikan nilai dari sebuah *opaque predicate*, maka predikat tersebut tidak memberikan pertahanan apapun pada perangkat lunak.

Algoritma Arboit merupakan algoritma *software watermarking* yang menggunakan *opaque predicate*. Arboit mengajukan dua jenis algoritma *watermark*, yaitu algoritma GA1 yang secara langsung menyisipkan *opaque predicate* dan *watermark* ke dalam program dan algoritma GA2 yang menyisipkan *watermark* melalui *opaque method*.

Pada GA1, untuk menyisipkan sebuah *watermark w*, *w* dibagi menjadi *k* bagian, yaitu w_0, w_1, ..., w_{k-1}, *k* titik cabang, b_0, b_1, ..., b_{k-1}, dipilih secara acak sepanjang aplikasi. Pada setiap titik percabangan b_i, *opaque predicate* tertentu ditambahkan ke dalam predikat dalam lokasi tersebut. Kumpulan bit *watermark* disisipkan melalui *opaqure predicate* yang terpilih. Di dalam *opaqure predicate*, bit-bit tersebut dapat dikodekan sebagai konstanta atau dengan memberikan sebuah pangkat pada setiap *opaque predicate*.

Untuk mengenali *watermark* tesrsebut, aplikasi dipindai untuk mengekstraksi semua *opaque predicate* yang teridentifikasi. Bit-bit dari *watermark* kemudian diuraikan dari *opaque predicate* tersebut.

Sebagai contoh, sebuah *watermark* dikodekan dalam dalam *opaque predicate* $x^2 \geq 0$. *Watermark* tersebut dapat disisipkan sebagai berikut:

```
class C {
   void m1(int a, int b) {
      ...
      if (a <= b) {...}
      else {...}
      ...
   }
```

↓

```
class C {
   void m1(int a, int b) {
      ...
      int x=1;
      if ((a <= b)&& (x*x >= 0)) {...}
      else {...}
      ...
   }
```

Gambar 4: Proses penyisipan dan pengekstraksian *watermark* pada algoritma GA1

Pada algoritma Arboit kedua, GA2, proses penyisipan dan pengekstraksian *watermark* serupa dengan GA1. Perbedaannya terletak pada penggunaan *opaque method* untuk menyisipkan *watermark*. *K* titik cabang b_0, b_1, ..., b_{k-1}, pertama kali dipilih secara acak sepanjang aplikasi. Untuk setiap titik cabang b_i, dibuat beberapa *opaque method*, kemudian pemanggilan *method* tersebut ditambahkan pada kode program.

Bit-bit *watermark* dikodekan dalam *opaque method* melalui *opaque predicate* yang dievaluasinya. Untuk mengenali *watermark*, aplikasi tersebut dipindai, sehingga dapat diekstraksi sejumlah *opaque predicate* yang dikenali melalu/i *signature* pada *opaque method*. Ketika sebuah kandidat *method* yang mungkin teridentifikasi, bagian isi dari *method* tersebut diperiksa untuk menemukan *opaque predicate* di mana *watermark* disisipkan. Berikut ini adalah ilustrasi bagaimana kode program mengalami transformasi oleh GA2:

```
class C {
   void m1(int a, int b) {
      ...
      if (a <= b) {...}
      else {...}
      ...
   }
```

↓

```
class C {
   boolean m2() {
      int x=1;
      return (x*x >= 0);
   }
   void m1(int a, int b) {
      ...
      if ((a <= b)&& m2()) {...}
      else {...}
```

```
    . . .
}
```

Gambar 5: Proses penyisipan dan pengekstraksian *watermark* pada algoritma GA2

Algoritma Arboit yang digunakan untuk perbandingan dalam makalah ini ialah implementasi algoritma tersebut dalam SandMark yang berupa algoritma *software watermarking* dinamis. Berbeda dengan GA1 dan GA2 yang bersifat statis, algoritma *dynamic* Arboit (DGA) menggunakan *state* pengeksekusian program untuk menyisipkan dan mengekstraksi *watermark*. Perbedaan DGA dengan dua algoritma sebelumnya ialah pada pemilihan titik percabangan. DGA memanfaatkan *execution trace* untuk mengidentifikasi himpunan titik percabangan *B*, bukan dengan pengambilan titik secara acak. Sifat dinamis algoritma ini meningkatkan kemampuan program untuk mengatasi serangan distortif.

Berikut ini adalah contoh 9 buah *opaque predicate* yang digunakan dalam DGA [3]:

$\forall x, y \in \mathbb{Z}$	$7y^2 - 1 \neq x^2$
$\forall x \in \mathbb{Z}$	$2 \mid \lfloor \frac{x^2}{2} \rfloor$
$\forall x \in \mathbb{Z}$	$2 \mid x(x+1)$
$\forall x \in \mathbb{Z}$	$x^2 \geq 0$
$\forall x \in \mathbb{Z}$	$3 \mid x(x+1)(x+2)$
$\forall x \in \mathbb{Z}$	$7 \nmid x^2 + 1$
$\forall x \in \mathbb{Z}$	$81 \nmid x^2 + x + 7$
$\forall x \in \mathbb{Z}$	$19 \nmid 4x^2 + 4$
$\forall x \in \mathbb{Z}$	$4 \mid x^2(x+1)(x+1)$

Gambar 4: *Opaque predicate* yang digunakan dalam implementasi algoritma *Dynamic* Arboit dalam SandMark

Meskipun tidak ada dari kesembilan *opaque predicate* di atas yang tahan secara kriptografis terhadap berbagai bentuk analisis, namun hal ini tidak mengurangi kehandalan algoritma tersebut untuk menyembunyikan *watermark* dalam perangkat lunak.

3. PERBANDINGAN

Software watermarking memiliki karakteristik khusus yang harus dipenuhi agar dapat diterapkan secara efektif. Kriteria utama yang harus dipenuhi oleh setiap algoritma *software watermarking* ialah penyisipan *watermark* tersebut tidak berpengaruh secara signifikan terhadap fungsionalitas dan kinerja perangkat lunak yang disisipi *watermark* dan ketahanannya terhadap berbagai serangan. Berikut ini adalah kriteria-kriteria evaluasi yang dapat digunakan untuk menilai algoritma *software watermarking* yang diajukan dalam kerangka kerja perbandingan algoritma *software watermarking* dalam [1]:

1. *Size efficiency* (*data rate / capacity*)
   Size efficiency memperhitungkan ukuran dari *watermark* yang disisipkan ke dalam perangkat lunak, sedangkan *data rate* ialah rasio antara program setelah disisipi *watermark* dengan program awal.
2. *Time efficiency*
   Waktu merupakan faktor yang kritis dalam mengevaluasi pengaruh penyisipan *watermark* pada aplikasi orisinal. Pengukuran waktu dapat dilakukan pada tiga tempat, yaitu waktu eksekusi program, waktu penyisipan, dan waktu pengekstraksian *watermark*.
3. Ketahanan terhadap serangan transformatif (*resilience*)
   Tujuannya ialah untuk mengukur sejauh mana algoritma *software watermarking* yang berbeda dapat tahan terhadap berbagai serangan yang mengubah kode sumber program.
4. Invisibilitas dan perseptibilitas (*stealthy*)
   Bertujuan untuk mengukur sejauh mana *watermark* yang disisipkan tersembunyi dari pengguna dan kesulitannya untuk dideteksi.
5. Sifat algoritma *watermarking*
   Menunjukkan kompleksitas algoritma yang digunakan dalam penyisipan dan ekstraksi *watermark*. Hal ini dipengaruhi oleh aplikasi yang disisipi *watermark*. Jika algoritma yang digunakan sangat kompleks dan aplikasi yang disisipi *watermark* berukuran sangat besar, maka harga yang harus dibayar untuk proses *watermarking* menjadi terlalu besar.
6. Faktor-faktor lain
   Faktor-faktor lainnya yang berpengaruh, seperti jenis informasi yang dapat disisipkan sebagai *watermark*, apakah hanya berupa angka atau string, dsb, ukuran kode sumber, jumlah kelas yang disisipi *watermark*, dan jumlah kunci yang dibutuhkan.

Dalam makalah ini, kriteria evaluasi yang digunakan ialah kriteria nomor 1 dan 3.

Perangkat lunak dan lingkungan pengujian

Pengujian dilakukan dengan menggunakan perangkat lunak khusus untuk studi algoritma *software watermarking*, yaitu SandMark 3.40 (Mystique), sedangkan spesifikasi lingkungan pengujian ialah sebagai berikut:

Tabel 1. Spesifikasi lingkungan pengujian

Framework	JDK 1.6.0
Sistem operasi	Microsoft Windows XP Professional Service Pack 2
CPU	Pentium 4 1.66 GHz
RAM	512 MB

Pengujian *size efficiency*

Pengujian dilakukan dengan menyisipkan *watermark*

kepada dua buah program aplikasi Java, kemudian dilihat perubahan ukuran program tersebut.

Tabel 2. Hasil pengujian *size efficiency*

Nama Program	Ukuran Semula (KB)	Ukuran Akhir (KB)		*Data rate*	
		CT	DGA	CT	DGA
TTT.jar	10.5	19.6	19.7	1.87	1.87

Pengujian ketahanan algoritma (*resilience*)
Pengujian ini dilakukan dengan melakukan tiga jenis serangan pada program aplikasi dan melihat apakah *watermark* yang disisipkan masih dapat dikenali atau tidak.

Tabel 3. Hasil pengujian *resilience*

Tipe serangan	Pengenalan *Watermark* (+/-)	
	CT	DGA
Code obfuscation	+	+
Code optimization	+	+
Collusive attack	+	+

Berdasarkan tabel di atas, dapat dilihat bahwa kedua algoritma tersebut berhasil mengekstraksi *watermark* yang disisipkan meskipun kode sumber telah mengalami perubahan akibat serangan yang dilakukan.

4. KESIMPULAN

Berikut ini adalah beberapa kesimpulan yang dapat diambil dari seluruh uraian di atas

1. *software watermarking* merupakan salah satu metode pengamanan perangkat lunak yang cukup efektif untuk memperbesar kemungkinan pembuktian tindakan pembajakan dan pelanggaran hak cipta pada perangkat lunak
2. terdapat dua macam kategori algoritma *software watermarking*, yaitu algoritma statis dan dinamis. Algoritma *watermarking* dinamis relatif lebih tahan terhadap serangan distortif dibandingkan algoritma statis
3. Algoritma CT merupakan algoritma *software watermarking* dinamis yang berbasis graf. Algoritma *Dynamic* Arboit adalah algoritma *software watermarking* yang menggunakan *opaque predicate*.
4. Terdapat beberapa kriteria evaluasi yang dapat digunakan untuk membandingkan algoritma *software watermarking*, antara lain *size efficiency*, *time efficiency*, *resilience*, *stealthy*, kompleksitas algoritma, jenis informasi *watermark*, jumlah kelas, dan ukuran kode sumber.
5. Berdasarkan pengujian *size efficiency* dan pengujian *resilience*, disimpulkan bahwa algoritma CT lebih baik dibandingkan dengan algoritma *dynamic* Arboit. Hal ini dilihat dari ukuran berkas program yang telah tersisipi *watermark* dengan algoritma CT yang lebih kecil dibandingkan dengan berkas program hasil algoritma *dynamic* Arboit.
6. Baik algoritma CT maupun algoritma *dynamic* Arboit, keduanya tahan terhadap tiga tipe serangan pada *software watermark*, yaitu *code obfuscation*, *code optimization*, dan *collusive attack*. Hal ini menunjukkan, algoritma *software watermarking* dinamis memiliki ketahanan yang cukup baik terhadap serangan distortif (*semantic-preserving transormation*).

DAFTAR REFERENSI

[1] Al-Dharrab, Mohannad Ahmad Abdulaziz, *Benchmarking Framework for Software Watermarking*, King Fahd University of Petroleum & Minerals, 2005, Dhahran, Saudi Arabia.

[2] Collberg, Christian, Andrew Huntwork, Edward Carter, dan Gregg M. Towsend, *Dynamic Graph-Based Software Watermarking*, Techical Report TR 04-08, Departement of Computer Science, University of Arizona, 2004, Tucson.

[3] Collberg, Christian S., dan Myles, Ginger, *Software Watermarking via Opaque Predicates*: *Implementation*, *Analysis*, *and Attacks*, Department of Computer Science, University of Arizona, Tucson

[4] Munir, Rinaldi, *Diktat Kuliah IF5054 Kriptografi*,Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung (STEI ITB), 2006, Bandung

[5] Rosadi, Febrian Aris, "Studi Tentang *Software Watermarking*", Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung (STEI ITB), 2006, Bandung