PENGGUNAAN *WATERMARKING* PADA PENYEBARAN *SOFTWARE* UNTUK PERLINDUNGAN HAK CIPTA

Amudi Sebastian – NIM : 13503017

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail: if13017@students.if.itb.ac.id

Abstrak

Pembajakan *software* dan pelanggaran hak cipta berkembang dengan pesat. Pada awalnya, penyebaran *software* membutuhkan pendistribusian perangkat fisik (misal: disket, cakram optikal), sehingga tingkat penyebaran *software* ilegal masih terbatas. Perkembangan pada teknologi jaringan dan internet menyebabkan peningkatan drastis pada penyebaran *software* ilegal. Oleh karena itu, perlu adanya metode untuk melindungi *software* dari pembajakan.

Metode yang umum digunakan adalah *watermarking*, yang biasanya digunakan untuk melindungi media digital dari pembajakan. Perlindungan terhadap *software* dapat dicapai dengan melekatkan suatu *watermark* tertentu yang keberadaannya dapat diverifikasi oleh pembuat *software*, sehingga pembuat dapat mengetahui siapa yang menggunakan *software* ilegal. *Software* yang disisipi *watermark* sangat mungkin akan mendapat serangan yang berusaha melacak keberadaan *watermark* dan menghapusnya, sehingga penyisipan *watermark* juga perlu memperhatikan hal tersebut. Sebuah salinan *software* juga harus dapat diverifikasi bahwa salinan tersebut belum pernah dimodifikasi oleh pihak lain, sehingga perlu diterapkan teknik-teknik tertentu untuk mendeteksi dan mencegah modifikasi/*tampering*.

Saat ini, algoritma *watermarking* dapat dideskripsikan sebagai statis dan dinamis. Makalah ini akan mencoba membahas tentang penggunaan *watermarking* untuk mencegah pembajakan *software*. Pembajakan *software* yang akan dibahas adalah berupa pembuatan salinan ilegal (*illegal copies*) dan *tampering attack* yang memodifikasi program untuk tujuan tertentu. Teknik *watermarking* yang dibahas adalah teknik *watermarking* statis (*static data watermark, code watermark)* dan dinamis (*Easter Egg watermark, dynamic data structure watermark, dynamic execution trace watermark, dynamic path-based watermark,* dan *dynamic graph watermark*).

***Kata kunci**: Watermarking, watermark, software watermarking, copyright, static software watermarking, dynamic software watermarking, tampering, software piracy.*

1 Pendahuluan

Perkembangan teknologi informasi, terutama dalam bidang jaringan komunikasi dan internet, telah membawa perubahan besar pada kehidupan sehari-hari. Penyebaran data dan informasi menjadi lebih mudah karena adanya media internet untuk distribusi, selain media digital konvensional yang sudah ada sebelumnya seperti CD, disket, dan sebagainya. Oleh karena itu, orang ingin melindungi haknya akan media digital tersebut. Salah satu caranya adalah dengan menyisipkan suatu pesan rahasia yang hanya pembuat media saja yang tahu. Teknik yang disebut *watermarking* ini sangat berguna untuk melindungi hak cipta pada media digital.

1.1 *Watermarking*

Watermarking – disebut juga tanda air – merupakan teknik untuk menyisipkan pesan rahasia kedalam pesan yang melindunginya [1]. Prinsip dasarnya mirip dengan *steganography* – yang memiliki arti pada bahasa asalnya adalah tulisan tersembunyi – dimana suatu pesan rahasia disembunyikan dalam suatu media sehingga pesannya tersamarkan tetapi medianya tetap jelas. Penggunaan *watermark* sudah sangat banyak, terutama pada media digital yang sarat akan hak cipta. Misalnya, penyebaran media dalam bentuk gambar, video, audio yang *copyrighted* memikiki *watermark* di dalamnya, baik yang dapat dilihat langsung maupun yang hanya diketahui pembuatnya. Contoh penggunaan *watermarking* adalah penyisipan informasi pembuat pada sebuah gambar digital. Dengan

watermark, kepemilikan terhadap suatu media digital dapat dibuktikan.

Watermark harus memiliki sifat-sifat atau properti tertentu agar dapat dimanfaatkan dengan baik. Sifat-sifat itu diantaranya:

1. *Resilient*, tidak mudah berubah. *Watermark* harus dapat bertahan terhadap serangan-serangan.
2. *Cheap*, murah untuk diimplementasi. *Watermark* tidak boleh memberikan *overhead* yang besar, tetapi harus seminimal mungkin.
3. *Stealthy*, tidak diketahui keberadaannya. *Watermark* harus dapat mempertahankan sifat-sifat statistic dari media penampungnya.
4. *Unique identifying property*, keberadaan *watermark* dapat dibuktikan dengan proses ekstraksi tertentu.

Untuk mendapatkan *watermark* yang baik, keempat sifat tersebut harus diperhatikan dan diimplementasikan. Tanpa adanya sifat-sifat diatas, *watermark* tidak akan dapat digunakan untuk membuktikan kepemilikan suatu media digital.

1.2 *Software Watermarking*

Software adalah salah satu dari teknologi yang paling berharga pada era informasi ini. *Software* digunakan untuk menjalakan berbagai perangkat, mulai dari *Personal Computer* sampai internet. Seiring dengan penggunaan *software* sebagai salah satu kakas penunjang produktivitas, penggandaan tidak sah terhadap *software* dan distribusinya – disebut juga pembajakan *software* – telah berlangsung secara global. Dibutuhkan suatu mekanisme untuk mengurangi pembajakan *software* yang telah terjadi, salah satunya dengan menggunakan *watermark*.

Sebelum *watermark* digunakan secara luas, ada beberapa teknik yang digunakan untuk melindungi *software* terhadap pembajakan [2], diantaranya:

1. *Hardware assisted*.

   Teknik ini melindungi *software* dengan bantuan berbagai perangkat keras yang kokoh, seperti *smart card*, *secure processors*, *hardware dongle*, dan *memory device* seperti *floppy disk*, CD-ROM. Teknik ini umumnya digunakan untuk *software high-end* yang didistribusikan dalam jumlah yang sedikit. Teknik ini tergolong sulit untuk diterapkan dan pada dasarnya tidak dapat mencegah *reverse engineering* dan *tampering* terhadap *software*.

2. Teknik berbasis *software*

   Ada beberapa teknik yang dapat diterapkan, diantaranya *obfuscation*, *watermarking*, *tamper-proofing*, *traitor sharing*, *secure evaluation*, dan *secret sharing*. Selain *watermarking* dan *tamper-proofing*, teknik-teknik diatas tidak akan dibahas dalam makalah ini.

Watermark umumnya digunakan untuk data multimedia digital. Selain itu, *watermark* juga dapat digunakan pada *software*, yang juga didistribusikan melalui media digital, untuk membuktikan keaslian *software* tersebut.

Pembajakan *software* semakin meningkat seiring dengan perkembangan teknologi. Penyebaran data digital semakin mudah dan cepat, data digital berupa gambar, video, musik, dan termasuk *software* menjadi sangat rentan akan pembajakan. Tentunya, pemilik hak cipta terhadap content atau *software* tersebut tidak ingin hasil karyanya dibajak dan digunakan oleh pihak-pihak yang tidak berhak.

Perlindungan terhadap data digital berupa gambar, video, dan audio menggunakan *watermark* sudah sangat banyak digunakan. Berbagai teknik tersedia untuk menyisipkan *watermark* kedalam media, dan berbagai *software* juga tersedia untuk mempermudah dan mempercepat prosesnya. Sedangkan, perlindungan terhadap *software* masih sangat minim. Meskipun telah banyak tulisan mengenai perlindungan terhadap pembajakan *software*, *software watermarking* merupakan area yang sangat jarang diperhatikan. Hal ini sangat tidak menguntungkan, mengingat bisnis *software* adalah bisnis dengan estimasi nilai lebih 15 miliar US dolar per tahun [3].

Ada tiga pokok persoalan yang harus diperhatikan dalam perancangan teknik *software watermarking*, diantaranya:

1. Perhitungan ukuran data yang dibutuhkan. Seberapa besar *watermark* yang akan disisipkan dibandingkan dengan ukuran *software*.
2. Bentuk dari *software* yang dilindungi. Apakah *software* akan didistribusikan dalam bentuk *native binary code* atau *virtual machine code*.

3. Model serangan yang diperkirakan. Serangan *de-watermarking* apa saja yang dapat diperkirakan.

1.3 Serangan Terhadap *Watermarking*

Untuk melindungi *software*, kita akan berasumsi bahwa pihak-pihak yang tidak berhak pasti akan selalu berusaha untuk membobol perlindungan yang diterapkan pada *software*. Untuk dapat mengimplementasi *watermark* yang baik, perancangan *watermark* juga harus memperhatikan dan sadar akan teknik-teknik serangan terhadap *watermark*. Secara umum, skema serangan terhadap *watermark* adalah menemukan *watermark*, mengubah atau menyimpangkan *watermark*, dan kemudian menghilangkan atau menghapus *watermark*.

Sebagai ilustrasi, akan digunakan skenario seperti berikut. Alice sebagai pemilik sah objek *O* menyisipkan *watermark W* dengan kunci *K*. Bob mencoba menjual kembali objek *O* milik Alice kepada Charlie, dengan mengaku bahwa objek *O* tersebut adalah miliknya. Sebelum Bob menjual *O*, dia harus terlebih dahulu memastikan bahwa *watermark* yang disisipkan oleh Alice telah dihapus atau dibuat menjadi tidak berguna. Kalau tidak, Alice dapat membuktikan bahwa dia adalah pemilik sah objek *O*, dan mengklaim bahwa hak atas kekayaan intelektualnya telah dilanggar.

Ada beberapa jenis serangan yang mungkin dilakukan terhadap *watermark*[4], diantaranya adalah:

1. Serangan transformasi.

   Serangan transformasi akan berusaha melakukan proses transformasi terhadap *software* dengan berbagai cara. Cara-cara yang mungkin dilakukan adalah kompilasi, optimisasi, *obfuscation*, dekompilasi, dan penghilangan *dead-code*. Serangan transformasi yang sukses akan dapat menghilangkan *watermark* dari *software* sasaran serangan. Oleh karena itu, *software* harus menerapkan teknik-teknik untuk melindungi dari serangan transformasi ini. Cara untuk melindungi *software* dari serangan jenis ini adalah dengan menyisipkan *watermark* di dalam struktur data, dan bukan di dalam struktur programnya.

2. Serangan substraktif

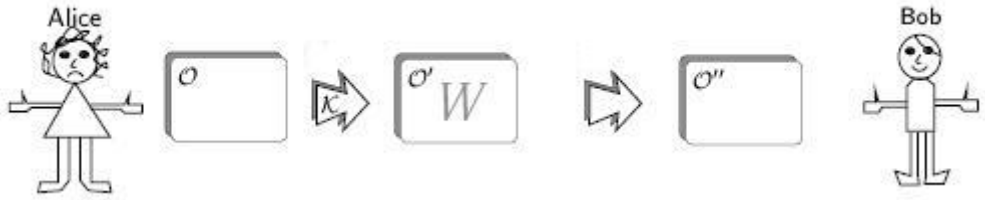
   Serangan substraktif adalah serangan terhadap *watermark* dimana pihak penyerang berusaha mencari dan menemukan lokasi dari *watermark* untuk kemudian dihapus dari *software* yang menjadi medianya. *Software* yang terkena serangan substraktif akan menjadi seperti *software* yang tidak memiliki *watermark*. Serangan substraktif yang sukses akan dapat mengambil *watermark* dari *software* sekaligus menghilangkannya. Sebagai ilustrasi, misalkan Alice memiliki objek *O*, dan menyisipkan *watermark W* dengan kunci *K* ke dalamnya, dan mendistribusikan objek baru, *O'*, tersebut. Bob melakukan serangan substraktif terhadap objek *O'*, dan berhasil menghapus *W* dari dalamnya, dan hasilnya adalah objek *O''* yang sama seperti *O*. Lihat Gambar 1.

3. Serangan distortif

   Serangan distortif adalah serangan yang dapat membuat *watermark* tidak terdeteksi lagi. Serangan distortif menerapkan transformasi distortif terhadap *software*, dan sekaligus terhadap *watermark* yang terkandung didalamnya. Serangan distortif dapat mengakibatkan penurunan dalam hal kualitas terhadap *software* sasaran. Namun, jika penurunan kualitas tersebut masih dapat diterima dan memiliki nilai guna tertentu oleh pihak penyerang, maka serangan distortif dapat dikatakan efektif. Sebagian besar teknik *media watermarking* rentan terhadap serangan distortif [5]. Sebagai ilustrasi, lihat Gambar 3.

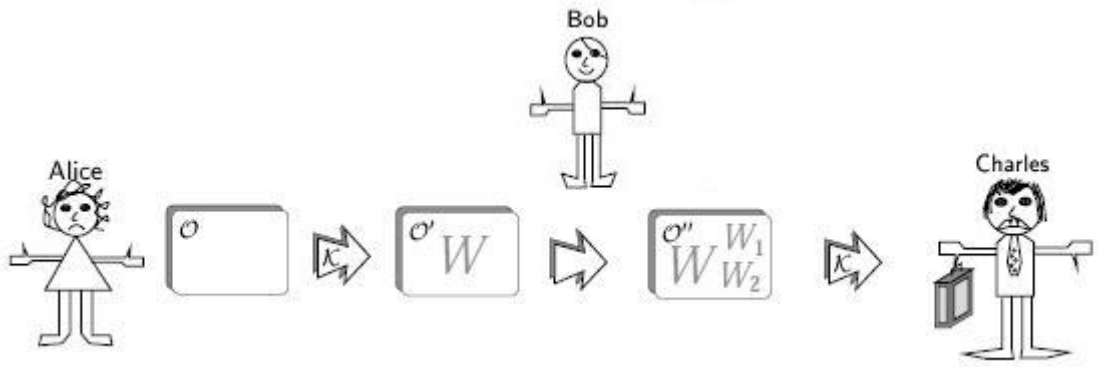
4. Serangan aditif

   Serangan aditif adalah serangan terhadap *software* yang mengandung *watermark* dengan cara memperbanyak *watermark* baru. Penyerang akan menambahkan *watermark* miliknya ke dalam *software* sasaran sehingga akan mencoba menyisipkan *watermark* ke dalam *software* sasaran dan kemudian mengekstraksinya kembali dari *software* tersebut, secara terus menerus. Tujuan dari serangan brute-force adalah untuk mempelajari cara mengekstraksi *watermark*, sehingga proses dilakukan terus menerus sampai akhirnya *watermark* asli dapat ditemukan. Karena sifatnya yang berulang-ulang tersebut, maka



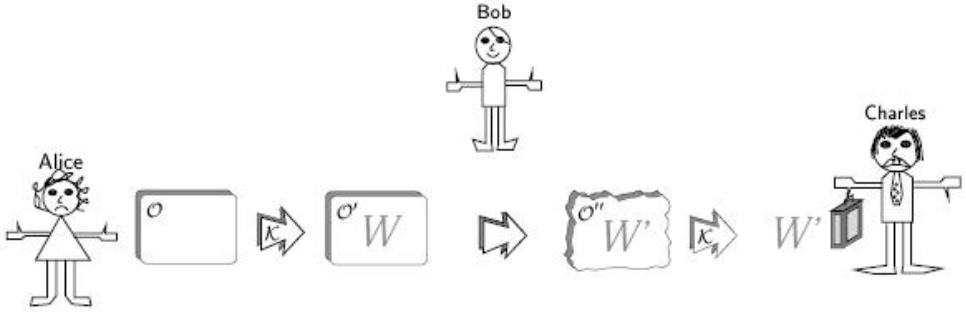
Gambar 1. Serangan Substraktif

Alice memiliki objek *O*, dan menyisipkan *watermark W* dengan kunci *K* ke dalamnya, dan mendistribusikan objek baru, *O'*, tersebut. Bob melakukan serangan substraktif terhadap objek *O'*, dan berhasil menghapus *W* dari dalamnya, dan hasilnya adalah objek *O''* yang sama seperti *O*.



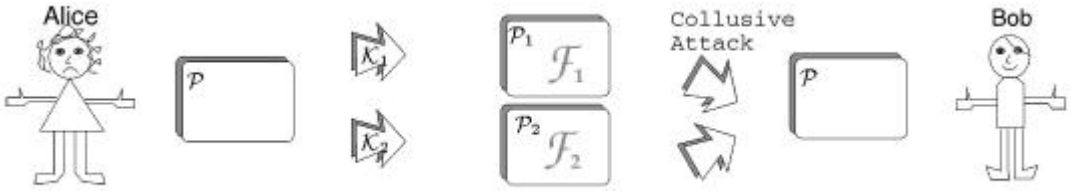
Gambar 2. Serangan Aditif

Alice memiliki objek *O*, dan menyisipkan *watermark W* dengan kunci *K* ke dalamnya, dan mendistribusikan objek baru, *O'*, tersebut. Bob melakukan serangan aditif terhadap *O'*, dengan cara menambahkan *watermark* miliknya sendiri, yaitu W_1 dan W_2, dan hasilnya adalah *O''*. Charles berusaha mengekstraksi *watermark* dari *O''* dengan kunci *K* dan tidak dapat menemukan *watermark W* milik Alice, sehingga Charles percaya bahwa *O''* adalah milik Bob.



Gambar 3. Serangan Distortif

Alice mem memiliki objek *O*, dan menyisipkan *watermark W* dengan kunci *K* ke dalamnya, dan mendistribusikan objek baru, *O'*, tersebut. Bob melakukan serangan distortif untuk melacak lokasi *W* dan kemudian menghapusnya, meskipun terjadi penurunan kualitas terhadap *O'*. Bob kemudian menyisipkan *watermark W'*, dan kemudian menjualnya kepada Charles. Charles mencoba mengekstraksi *watermark* dari *O''* hanya akan menemukan *W'* yang menyatakan bahwa objek *O''* merupakan buatan Bob.



Gambar 4. Serangan Kolusi
Alice memiliki *software P*, dan terdapat dua salinan. Keduanya masing-masing diberi *fingerprint* yang berbeda, F_1 dan F_2. Bob melakukan serangan kolusif dengan membandingkan kedua salinan yang masing-masing memiliki *fingerprint* yang berbeda, sehingga Bob dapat menghasilkan salinan *software P* tanpa *fingerprint* milik Alice.

biasanya serangan jenis ini membutuhkan waktu yang lama.

1.4 Hak Cipta

Hak cipta adalah sekumpulan hak-hak eksklusif yang diberikan oleh pemerintah untuk mengatur penggunaan hasil penuangan gagasan atau informasi tertentu. Dengan kata lain, hak cipta adalah hak untuk menyalin dan memperbanyak suatu ciptaan. Hak cipta dapat memungkinkan pemegang hak tersebut untuk membatasi penggandaan dan penyebaran secara tidak sah terhadap suatu ciptaan.

Setiap negara memiliki undang-undang tersendiri yang khusus untuk mengatur tentang hak cipta ini. Di Indonesia, peraturan perundang-undangan yang mengatur masalah hak cipta dijelaskan pada [14]. Menurut undang-undang tersebut, pengertian hak cipta adalah "hak eksklusif bagi pencipta atau penerima hak untuk mengumumkan atau memperbanyak ciptaannya atau memberikan izin untuk itu dengan tidak mengurangi pembatasan-pembatasan menurut peraturan perundang-undangan yang berlaku."

Hak cipta berlaku pada berbagai jenis karya seni atau karya ciptaan. Ciptaan tersebut dapat meliputi puisi, drama, serta karya tulis lainnya, film, karya-karya koreografis seperti tari, balet, dan sebagainya, komposisi musik, rekaman suara, lukisan, gambar, patung, foto, siaran radio dan televisi, desain industri, dan tentunya *software* komputer.

2 Teknik *Watermarking*

Setelah mempelajari jenis-jenis dan teknik-teknik serangan diatas, maka dapat disimpulkan bahwa tidak ada suatu teknik *software watermarking* yang dapat mengatasi seluruh jenis serangan. Ada berbagai macam teknik *watermarking* yang telah diciptakan, tetapi setiap teknik *watermarking* memiliki kelebihan dan kekurangannya masing-masing. Secara garis besar, teknik *watermarking* dibagi menjadi dua bagian, statis dan dinamis. Secara umum, perbedaannya adalah letak dan penempatan *watermark*, apakah diletakkan di dalam bagian code atau data dari *native executable* atau *class file*, atau diletakkan dalam struktur *run-time* program.

2.1 *Static Software Watermarking*

Static software watermarking adalah teknik dimana *watermark* yang akan disisipkan ke dalam *software* ditempatkan di dalam *executable software* tersebut. Misalnya dalam lingkungan Unix, biasanya *watermark* dapat ditempatkan dalam bagian *initialized data* (tempat dimana string statis disimpan), di dalam bagian *text* (code *executable*), atau di bagian *symbol* (informasi *debugging*). Misalnya pada lingkungan Java, informasi dapat disembunyikan dalam bagian manapun dari format file *class*, yaitu *constant pool table, method table, line number table*, dan sebagainya.

Contoh penerapan *static software watermarking* dalam *software*:

```
CONST C = "Copyright (C) ........."
```

Static Data Watermark, watermark disisipkan pada nilai konstanta dalam program.
Watermark akan disimpan dalam *initialized data section* dari program.

```
char V;
switch e {
```

```
    case 1: V = 'C';
    case 5: V = 'O';
    case 7: V = 'P';
    case 9: V = 'Y';
    case 10: V = 'R';
    ..............
}
```

Code Watermark, *watermark* disisipkan pada bagian kode program.

Untuk membedakan *static software watermarking*, maka pengelompokan akan dibagi menjadi dua, yaitu *code watermark* dan *data watermark*. *Code watermark* menyisipkan informasi *watermark* ke dalam bagian dari *executable software* yang mengandung instruksi-instruksi program. Sedangkan *data watermark* menyisipkan informasi *watermark* ke dalam bagian lainnya, seperti *header*, konstanta string, *debug information*, dan sebagainya.

2.1.1 *Static Data Watermark*

Data *watermark* sudah sangat umum karena sangat mudah implementasinya. Sebagai contoh, pemberitahuan hak cipta grup JPEG dapat dengan mudah diekstraksi dari distribusi *binary software Netscape*:

```
#strings  /usr/local/bin/netscape
\
| grep –i copyright
  Copyright  (C)1995,  Thomas  G.
  Lane
```

Static data watermark ditempelkan ke dalam program saat kompilasi, dan tidak akan berubah pada saat eksekusi program. Contoh *static data watermark* misalnya:

- `const char c[] = "Copyright (c) ...";` atau
- `const struct wmark= {0x12, 0x34, ...};`

Cara untuk mengekstraksi *static data watermark* adalah dengan mengetahui lokasi *watermark* dalam program, dan mengetahui apa atau bagaimana *watermark* yang ingin dicari. Orang yang membuat *watermark* dan menyisipkannya ke dalam program tertentu tentu akan dengan mudah mengekstraksi *watermark*, dan memverifikasi bahwa program tersebut adalah buatannya.

Watermark yang dapat disisipkan dalam *software* bukan hanya teks atau string tertentu saja. Untuk mempersulit proses pelacakan *watermark* dan proses ekstraksi *watermark* oleh penyerang, dan memperkokoh *watermark* terhadap serangan, *watermark* dapat menggunakan data gambar atau media digital lainnya. Metode ini adalah salah satu variasi *static data watermarking*. Caranya adalah dengan dengan cara menyisipkan *watermark* ke dalam gambar atau media digital lainnya (audio, video), dengan menggunakan salah satu algoritma *media watermarking* yang umum, dan kemudian gambar tersebut disimpan di dalam bagian *static data* dari program. Penyerang yang berhasil menemukan lokasi *watermark* dalam bagian *static data* dari program tidak akan langsung dapat mengetahui *watermark* tersebut, karena yang disimpan dalam bagian *static data* hanyalah data digital penampung *watermark* sesungguhnya. Untuk mendapatkan *watermark* yang asli harus melalui proses ekstraksi *watermark* dari media (gambar, video, audio) tersebut.

Sayangnya, *static data watermark* sangat rentan terhadap serangan distortif, terutama dengan penerapan *obfuscation*. Sebagai contoh, *obfuscator* sederhana terotomatisasi mungkin akan memecah semua string dan data statis lainnya menjadi potongan-potongan lebih kecil dan kemudian menyebarkannya dalam *executable* tersebut. Hal ini membuat proses ekstraksi dan pengenalan *watermark* menjadi mendekati mustahil.

Menurut Collberg [7], terdapat teknik untuk menghilangkan *watermark* dengan lebih elegan, yaitu dengan mengkonversi seluruh data statis dari program menjadi sebuah program sendiri yang memproduksi data tersebut.

2.1.2 *Code Watermark*

Code watermark adalah teknik penyisipan *watermark* dengan cara menyisipkannya dalam bagian *executable software* yang mengandung instruksi-instruksi program.

Ada tiga skema yang umumnya dipakai dalam code *watermark*. Yang pertama adalah *watermark* bergantung pada karakteristik dan urutan kode program, misalnya urutan dari statement *case* dalam program atau urutan statement-statement dalam program yang tidak saling bergantung secara fungsional. Yang kedua adalah *watermark* yang bergantung pada urutan penggunaan *register* pada *stack*, yaitu urutan pemanggilan *push* dan *pop* terhadap register. Yang ketiga adalah *watermark* yang menggunakan *control graph* dari program tersebut.

Watermark pada media digital seperti gambar, audio, dan video pada umumnya disisipkan pada bit yang berlebih, yaitu bit yang tidak terdeteksi karena keterbatasan persepsi manusia (*least significant bit*). Code *watermark* dapat dikonstruksi melalui cara yang serupa, karena objek kode dapat mengandung informasi yang berlebih. Misalnya, jika tidak ada kebergantungan data maupun kontrol pada dua statement yang bersisian *S1* dan *S2*, maka mereka dapat dipertukarkan posisinya. Bit *watermark* kemudian dapat di *encode* dalam penempatan statement, apakah *S1* dan *S2* berada dalam urutan yang sesuai urutan yang ditentukan atau tidak.

Terdapat banyak variasi teknik seperti diatas. Contohnya IBM, ketika menuntut pembajak *software* yang telah menduplikasi PC-AT ROM mereka, IBM mengatakan bahwa urutan *register* yang di *push* dan *pop* menyatakan *signature* dari *software* mereka. Dengan prinsip yang sama, dengan mengurutkan percabangan dari *m-branches case statement*, kita dapat meng-*encode* $\log_2(m!) \approx O(m \log m)$ bit *watermark*.

Code watermark juga dapat digunakan untuk meng-*generate* dan meng-*encode serial number* untuk *software* tersebut [8]. Serial number *software* akan di-*encode* dalam sekuens blok dasar dari graf *control flow* program. Pemilik *software* yang akan mendistribusikan *software*nya akan memberikan *serial number* – nomor seri – bagi tiap orang yang secara legal memiliki hak untuk memiliki salinan *software* tersebut. Jika pihak yang tidak berhak berusaha menyalin *software* tersebut, maka ia tidak akan dapat menggunakan *software* tersebut karena tidak memiliki *serial number*.

Banyak *code watermark* yang rentan terhadap serangan distortif sederhana. Misalnya, metode dari [8] dapat dengan mudah dihancurkan dengan melakukan *locality-improving optimization* yang banyak. Metode itu juga tidak menyediakan perlindungan terhadap serangan aditif. Jika penyerang struktur blok dasar dari program untuk meng-*encode watermark* miliknya, maka jelas bahwa *watermark* asli tidak dapat diketahui keberadaannya kembali.

Beberapa teknik *code obfuscation* yang juga dapat menggagalkan pengenalan *code watermarking* dengan sukses. Blok dasar dapat dengan mudah dihancurkan hanya dengan menambahkan percabangan yang selalu bernilai benar. Sebagai contoh:

```
void P() {
    S1;
    S2;
    S3;
    ...
}
```

Dapat ditransformasi menjadi:

```
void P() {
    S1;
    if (P^T) S2;
    S3;
    ...
}
```

dan *watermark* yang tersimpan didalamnya menjadi tidak dapat diekstraksi kembali.

2.1.3 *Tamper-proofing Static Watermark*

Pengalaman dengan *obfuscation* mengatakan bahwa seluruh struktur statis dari sebuah *software* dapat diacak dengan mudah dengan teknik serangan transformasi yang memanfaatkan *obfuscation*. Pada kasus dimana *obfuscation* terlalu mahal untuk dilakukan, maka *inlining* dan *outlining*, dan berbagai macam teknik transformasi *loop* dan *code motion* adalah teknik optimasi yang terkenal dan dapat dengan mudah menghancurkan *static code watermark*.

Cara untuk melindungi *static* code *watermark* adalah dengan membuat *watermark* tersebut *tamper-proof* atau tahan terhadap kerusakan. Contohnya, Moskowitz [28] menggambarkan bagaimana metode *software watermarking* mereka *tamper-proof*.

Selain menggunakan data gambar untuk menampung *watermark*, dalam data gambar tersebut juga disisipi potongan kode esensial. Pada rentang waktu tertentu, potongan kode ini diambil, diekstraksi dari data gambar, dan kemudian dieksekusi. Jika kode tidak ditemukan, atau kode telah berubah, dan data gambar serta *watermark* telah berubah, maka *software* dapat dibuat agar tidak dapat berjalan lagi. Namun, membangkitkan dan mengeksekusi kode seperti itu dapat dianggap kelakuan yang tidak normal untuk kebanyakan *software*.

Kesulitan selanjutnya untuk membuat *software tamper-proof* adalah kesulitan *tamper-proofing*

kode *watermark* terhadap jenis-jenis transformasi yang mempertahankan semantik dari kode. Hal ini berlaku misalnya pada Java. Program Java tidak dapat menginspeksi kode program itu sendiri. Dalam bahasa lain, misalnya C, program masih dapat melakukan hal tersebut, namun sangat tidak normal karena program memeriksa kode program dan bukannya segmen data dari program yang berjalan.

Oleh karena itu, meskipun memiliki tingkat kesederhanaan dan popularitas yang tinggi, *static software watermarking* dapat dianggap tidak aman dan memiliki kekurangan.

2.2 *Dynamic Software Watermarking*

Seperti yang telah dibahas sebelumnya, *watermark* statis sangat mudah diserang dengan serangan transformasi dengan teknik transformasi yang mempertahankan semantik. Selanjutnya akan dibahas mengenai *dynamic software watermarking*.

Watermark dinamis berbeda dengan yang statis. Jika *watermark* statis menyimpan *watermark* dalam kode program, sebaliknya *watermark* dinamis menyimpan *watermark* dalam *execution state* dari program. Hal ini membuat sebagian dari *watermark* dinamis lebih mudah dibuat menjadi *tamper-proof* terhadap serangan transformasi yang memanfaatkan *obfuscation*.

Pada *dynamic software watermarking*, pengguna *software* akan mengeksekusi input yang berurutan, dengan urutan tertentu, dan kemudian *software* akan berada dalam keadaan dimana *watermark* direpresentasikan. Hal inilah yang membedakan dengan *static software watermarking*, karena hanya dengan urutan input tertentu saja *watermark* dapat diketahui keberadaannya.

Ada beberapa jenis *dynamic software watermark*. Pada pembahasan setiap jenis *watermark*, diasumsikan *software O* dieksekusi dengan sekuens input yang telah ditentukan sebelumnya, yaitu $I = I_1...I_k$, yang membuat *software* memasuki keadaan dimana *watermark* direpresentasikan dan dapat diekstraksi atau diketahui keberadaannya. Berikut ini adalah pembahasan setiap jenis *dynamic software watermarking*, yang masing-masing metode memiliki teknik-teknik tertentu yang berbeda-beda.

2.2.1 *Easter Egg Watermark*

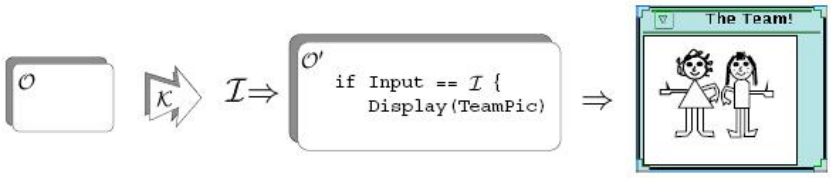
Easter Egg watermark, seperti dapat dilihat pada Gambar 5, adalah teknik penyisipan *watermark* pada *software*, dimana *watermark* dapat berupa fungsionalitas tertentu. Misalnya, dengan sekuens input tertentu yang tepat, *software* akan mengeluarkan informasi *copyright* yang tersimpan di dalamnya. Potongan kode program akan dieksekusi jika pengguna *software* berhasil memasukkan urutan kode tertentu yang ditentukan oleh pembuat *software*. Biasanya sekuens input tersebut sangat tidak biasa, dan sangat spesifik terhadap *software* tertentu saja.

Easter Egg sangat populer di kalangan pengembang *software*. Sangat banyak *Easter Egg* yang telah berhasil ditemukan, baik ditemukan oleh pengguna *software* maupun dipublikasikan sendiri oleh pengembang *software* tersebut. Misalnya, jika mengetikkan `about:credits` pada *browser* Mozilla Firefox, maka *browser* akan menampilkan para kontributor mereka, yaitu orang-orang yang telah ikut berkontribusi pada pembuatan *software browser* Mozilla Firefox tersebut. Hasilnya adalah seperti pada Gambar 9.

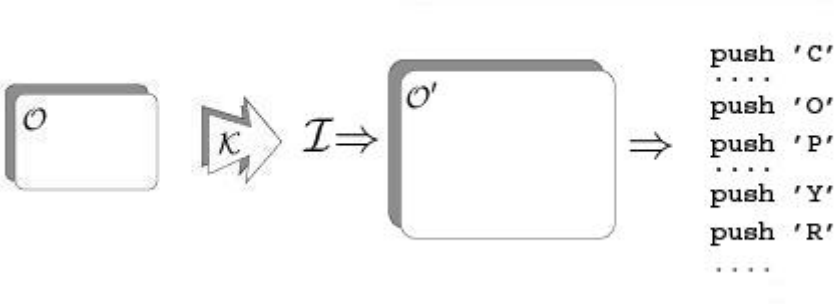
Contoh *software* lain yang menyisipkan *Easter Egg* adalah Bloodshed Dev-C++. *Software* ini menyisipkan *watermark* yang dapat dilihat dengan cara melakukan sekuens input tertentu, yaitu klik menu *Help*, kemudian klik *About Dev-C++..*, kemudian klik dan *drag* logo Dev-C++ di kiri atas *dialog box* tersebut dan *drop* di tombol Authors dibawahnya. Maka akan muncul gambar ikan kuning yang bergerak dari kiri ke kanan, dan jika di klik akan berubah arah, seperti pada Gambar 10.

Selain *watermark* mengenai pembuat dan *copyright*, beberapa *software* juga menggunakan *watermark* untuk tujuan lain. Misalnya, pada Mozilla Firefox, jika pengguna mengetikkan input berupa `about:mozilla` pada browser tersebut, maka akan muncul halaman web tertentu seperti Gambar 11. Pembuatnya menyebutnya sebagai *book of Mozilla*, yang merupakan metafora mengenai sejarah Mozilla. *Easter Egg* ini sangat populer di kalangan penggunanya, bahkan beberapa orang membuat dan menambahkan *book of Mozilla* karangannya sendiri dan mempublikasikannya. *Easter Egg* ini sama sekali tidak berhubungan dengan peringatan hak cipta, dan hanya sekadar sisipan biasa tanpa tujuan peringatan hak cipta apapun.

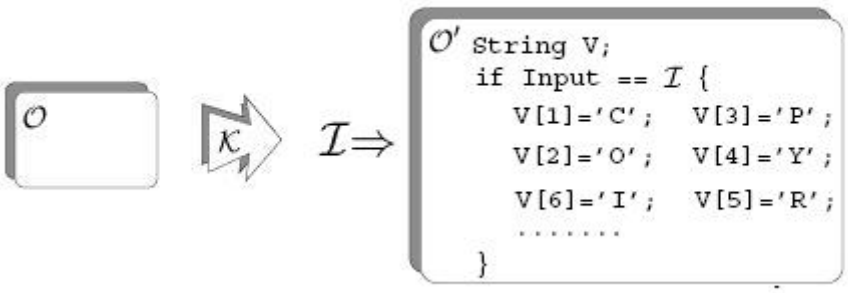
Permasalahan utama pada *Easter Egg* adalah *Easter Egg* mudah untuk ditemukan. Bahkan ada beberapa situs web tersendiri yang



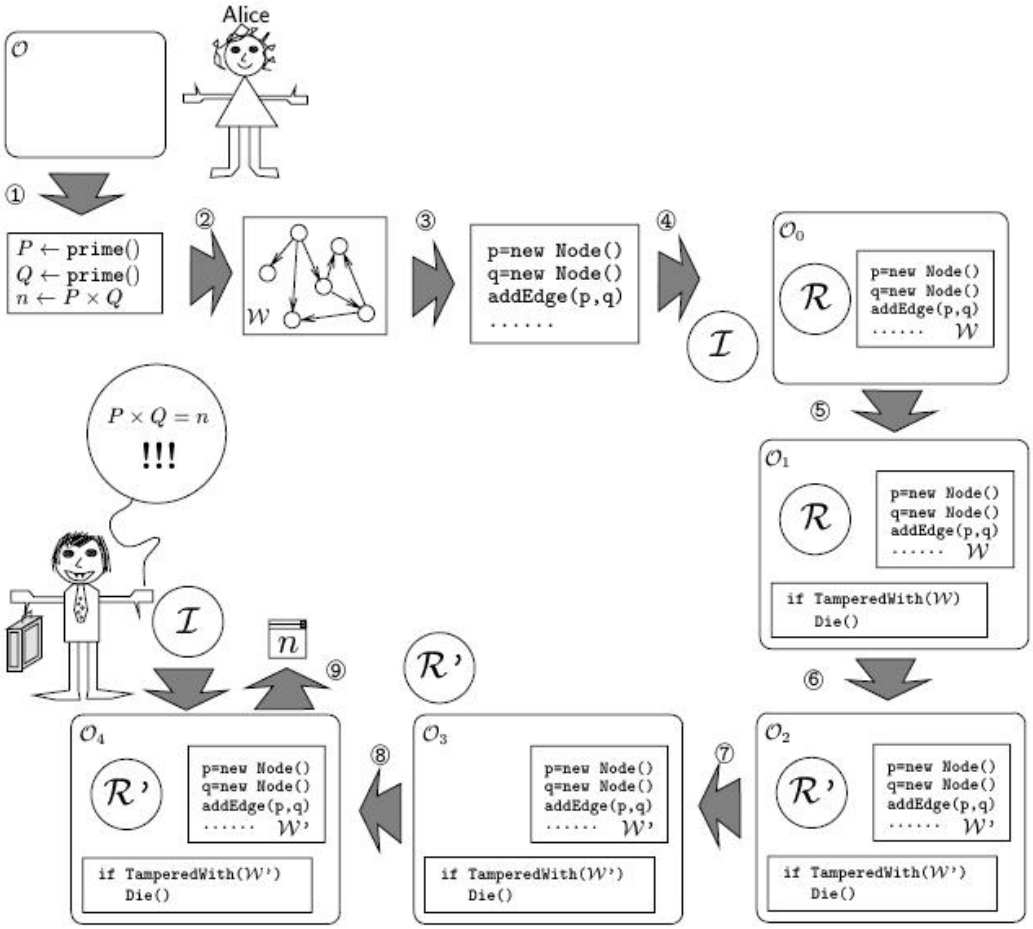
Gambar 5. Skema *Easter Egg Watermark*



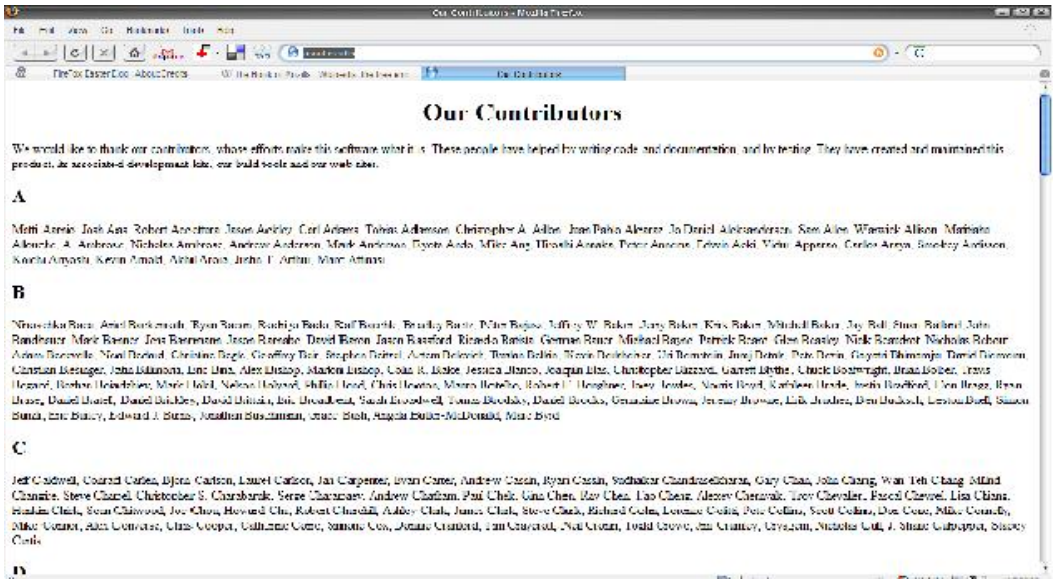
Gambar 6. Skema *Dynamic Execution Trace Watermark*



Gambar 7. Skema *Dynamic Data Structure Watermark*

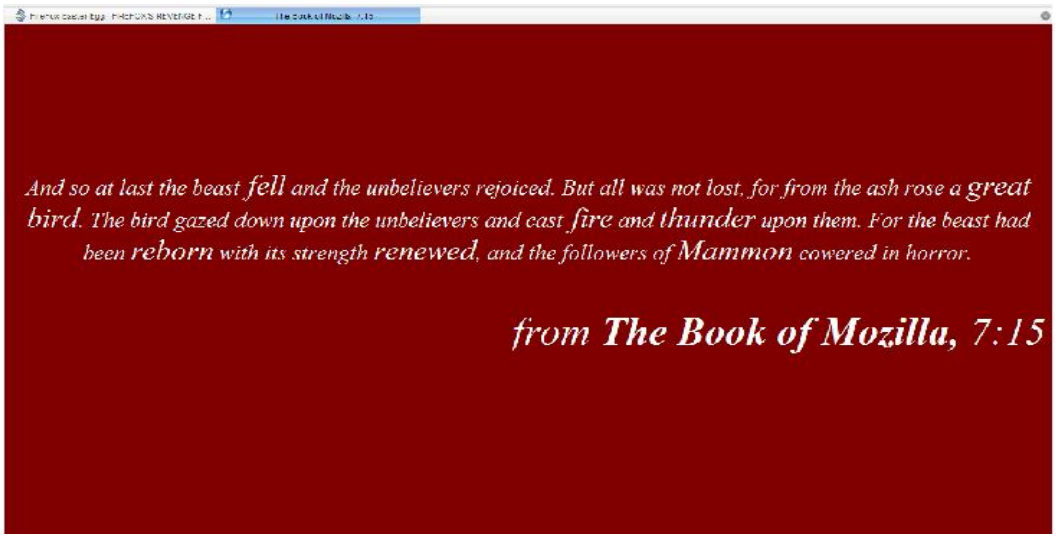


Gambar 8. *Dynamic* Graph *Watermark*



Gambar 9. *Easter Egg* pada Browser Mozilla Firefox

Gambar 10. *Easter Egg* pada *Software* Bloodshed Dev-C++



Gambar 11. *Easter Egg* pada Browser Mozilla Firefox

```
(1) bool A,B,C;          (1') short a1,a2,b1,b2,c1,c2;
(2) B = False;           (2') b1=0; b2=0;
(3) C = False;     T     (3') c1=1; c2=1;
(4) C = A & B;     ⇒     (4') x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(5) C = A & B;           (5') c1=(a1 ^ a2) & (b1 ^ b2); c2=0;
(6) if (A) ···;          (6') x=2*a1+a2; if ((x==1) || (x==2)) ···;
(7) if (B) ···;          (7') if (b1 ^ b2) ···;
```

Gambar 12. Contoh Kode Program yang Memecah Variabel

mengumpulkan hasil penemuan *Easter Egg* tersebut [10], baik dari *software*, film, games, *hardware*, musik, dan sebagainya. Begitu sekuens input yang benar berhasil ditemukan, *Easter Egg watermark* dapat dengan mudah ditemukan lokasinya dan kemudian dengan teknik *debugging* standar dapat dihapus secara sempurna.

2.2.2 *Dynamic Data Structure Watermark*

Gambar 7 menunjukkan skema *dynamic data structure watermark*. Pada skema tersebut, *watermark* disisipkan dalam *state* (*global, heap, stack data*, dan sebagainya) dari program ketika program dijalankan dengan input *I* tertentu. *Watermark* dapat diekstraksi dengan memperhatikan nilai yang disimpan dalam variabel-variabel *O*, setelah sekuens input *I* telah berakhir. Ekstraksi *watermark* dapat dilakukan dengan membuat fungsi tersendiri dalam program yang dirancang khusus untuk mengekstraksi *watermark*, atau dengan menjalankan program dalam *debugger*.

Dynamic data structure watermark memiliki beberapa sifat khusus yang menarik. Karena tidak ada output yang dihasilkan, penyerang tidak dapat mengetahui *watermark* secara langsung ketika sekuens input tertentu dimasukkan. Hal ini sangat berkebalikan dengan *Easter Egg watermark*, dimana output akan segera keluar sesaat setelah sekuens input tertentu dimasukkan. Pada *Easter Egg watermark*, ecara teori penyerang dapat membangkitkan sekuens input acak secara otomatis, dan menunggu sampai output yang berbeda dari biasanya muncul. Hal ini tidak dimungkinkan dalam *dynamic data structure watermark*. Sebagai tambahan, dalam *dynamic data structure watermark*, informasi mengenai *watermark* yang tersimpan dan lokasi *watermark* dalam *executable* program sangat sedikit, karena fungsi untuk mengenali *watermark* tidak disertakan dalam *software*.

Dynamic data structure watermark juga memiliki kekurangan. *Dynamic data structure watermark* juga rentan terhadap serangan transformasi dengan teknik *obfuscation*. Ada beberapa teknik *obfuscation* yang dapat dipergunakan untuk menghancurkan *dynamic data structure watermark*. Collberg menunjukkan pada [7, 11] bahwa satu variabel dapat dipecah menjadi beberapa variabel, dan demikian juga sebaliknya, beberapa variabel dapat digabungkan menjadi satu variabel. Gambar 12 merupakan contoh kode yang memecah satu variabel menjadi beberapa variabel. Teknik transformasi lain yang juga dapat digunakan adalah penggabungan/pemisahan *array*, modifikasi hirarki pewarisan pada *software object-oriented*, dan sebagainya.

2.2.3 *Dynamic Execution Trace Watermark*

Gambar 6 menunjukkan *skema dynamic execution trace watermark*. *Watermark* disisipkan dalam *trace* program, entah itu dalam instruksi program atau alamat memori atau keduanya, ketika program tersebut dieksekusi dengan masukan berupa sekuens input *I* tertentu. *Watermark* tersebut dapat diekstraksi dengan memperhatikan beberapa properti statistikal dari *address trace* dan/atau sekuens operator yang dieksekusi. Pembuat program dapat merancang agar programnya memiliki urutan eksekusi instruksi tertentu, sehingga urutan eksekusi tersebut sifatnya unik untuk program tertentu.

Beberapa transformasi yang dapat digunakan untuk meng-*obfuscate* kode juga akan melakukan hal yang sama terhadap *instruction trace*. *Watermark* dapat dengan efektif dihapus dari *software* dengan serangan transformasi. Transformasi yang lebih potensial dan lebih mahal dalam hal biaya komputasi adalah transformasi yang mengkonversi suatu bagian kode (misalnya Java *bytecode*) menjadi kode *virtual machine* yang lain yang benar-benar berbeda. Kode baru tersebut kemudian dieksekusi secara tersendiri. Perilaku kedua kode program tersebut akan sama persis, tetapi keduanya akan menghasilkan *execution trace* yang sama sekali berbeda. Biasanya cara ini tidak dilakukan karena cara ini tidak praktis dan memerlukan proses interpretasi tambahan di awal.

2.2.4 *Dynamic Path-Based Watermark*

Dynamic path-based watermark adalah teknik *watermarking* dimana *watermark* disisipkan dalam struktur percabangan dalam program. Teknik ini memiliki beberapa konsekuensi yang menarik. Pertama, percabangan maju yang dipilih oleh program adalah bagian esensial dari sebuah program yang membuat suatu program unik. Kedua, percabangan secara alami bersifat biner, sehingga mudah untuk menyisipkan *string* bit. Ketiga, *dynamic path-based watermark* dapat memudahkan *error-correction* dan *tamper-proofing*. Keempat, percabangan terdapat di mana-mana dalam program, sehingga *dynamic path-based watermark* tidak dapat dikenali menggunakan serangan statistikal.

Sebuah contoh sederhana dari *dynamic path-based watermark* adalah sebagai berikut. Program asli:

```
int  main(int argc) {
    if (argc == 3)
    printf("Rahasia");
    return 0;
}
```

dan penyisipan *watermark* berupa *bit string* 01010110 pada program asli adalah sebagai berikut:

```
int  main(int argc) {
    int a = 1, b = 0;
    if (argc == 3) {
        if (b == 0) a = 0;
        if (b != 0) a = 0;
        if (b == 0) a = 0;
        if (b != 0) a = 0;
        if (b == 0) a = 0;
        if (b != 0) a = 0;
        if (b != 0) a = 0;
        if (b == 0) a = 0;
        printf("Rahasia");
    }
    return 0;
}
```

Pada program diatas, sekuens input yang harus dilakukan adalah mengeksekusi program dengan jumlah argumen sebanyak 3, dan kemudian kode yang mengandung *watermark* akan dieksekusi.

Berbagai teknik penyisipan *watermark* dalam kode percabangan, serta teknik *tamper-proofing* untuk memperkuat kode terharap serangan transformasi dibahas lebih mendetail pada [12].

2.2.5 *Dynamic* Graph *Watermark*

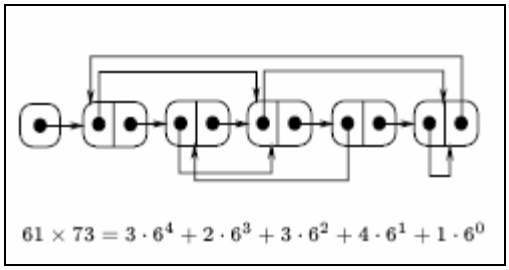
Seperti yang telah dibahas diatas, seluruh jenis teknik *software watermarking* rentan terhadap serangan distortif dan serangan transformatif yang mempertahankan semantik program. Oleh karena itu, teknik baru dirancang agar *watermark* dapat disisipkan dalam topologi dari struktur graf dinamis. Kode yang memanipulasi struktur graf dinamis akan sulit dianalisis karena efek pointer. Oleh karena itu, serangan transformasi yang mempertahankan semantik program yang melakukan perubahan mendasar terhadap graf akan sulit untuk dilakukan, sehingga kode dengan *dynamic graph watermark* lebih mudah dibuat *tamper-proof* dibandingkan dengan melakukan *tamper-proofing* terhadap kode atau data skalar.

Sebagai ilustrasi, lihat Gambar 8. Tahapan-tahapan dalam implementasi *dynamic graph watermark* adalah:

1. Alice memilih dua bilangan prima besar, dan menghitung hasil perkaliannya *n*. Kemudian *n* disisipkan dalam topologi graf. Graf tersebut adalah *watermark* milik Alice.
2. Kemudian *watermark W* tersebut diubah menjadi sebuah program yang membentuk graf yang sesuai. Program ini kemudian disisipkan ke dalam *software* asli *O*, sehingga jika *O* dieksekusi dengan input berupa sekuens tertentu, *W* akan dibentuk.
3. Program *R* yang berguna untuk mengenali dan mengekstraksi *watermark* juga dapat dibentuk dan disisipkan ke dalam *O*.
4. *Tamper-proofing* diterapkan kepada *O* untuk mencegah penyerang untuk dapat mengubah graf menjadi bentuk yang tidak dapat dikenali oleh *R*. Selanjutnya, *obfuscation* diterapkan pada *software* tersebut sehingga serangan *pattern-matching* tidak dapat diterapkan terhadap *O*.
5. Sebelum *O* didistribusikan, program pengenal *watermark R* perlu dihilangkan dari paket *software* tersebut.

Hasilnya adalah *software* milik Alice yang mengandung *watermark* berupa graf serta program pengenal *watermark R*. Charles dapat mengetahui bahwa *software* tersebut milik Alice dengan menghubungkan *software* tersebut kepada *R*, dan memasukkan sekuens input tertentu. Charles akan dapat memfaktorkan *n* yang dihasilkan oleh *R* menjadi dua bilangan prima yang dipilih oleh Alice saat pembuatan *watermark*.

Salah satu teknik sederhana untuk menyisipan graf ke dalam program adalah Radix-*k encoding*. Gambar 13 adalah salah satu contoh ilustrasi penggunaan teknik Radix-*K* untuk menyisipkan struktur graf ke dalam program.



Gambar 13. Radix-6 *encoding*

Pada gambar tersebut, pointer kanan akan menyimpan informasi *next*, dan pointer kiri meng-*encode* digit *base-k*.

3 *Tamper-proofing*

Tamper-proofing adalah teknik pencegahan terhadap perusakan *software* yang dapat menyebabkan perubahan kelakuan dalam *software*. Misalkan suatu *software S* didistribusikan untuk umum. Pembuatnya ingin mencegah seseorang untuk mengeksekusi *software* tersebut jika ternyata *software* tersebut telah mengalami modifikasi. Modifikasi tersebut dapat dilakukan oleh pihak mana pun, karena *S* tersedia untuk umum dan setiap orang dapat memperoleh salinannya. *Software S* seharusnya tidak dapat dieksekusi jika *software* tersebut mengandung *watermark* dan kode yang membentuk *watermark* telah dimodifikasi, atau *S* telah disisipi *virus*, atau *S* adalah *software e-commerce* dimana kemananan adalah bagian yang sangat penting dan sebagian kode *S* telah dimodifikasi.

Serangan modifikasi tersebut disebut *tampering attack*, dan teknik untuk mencegahnya disebut *tamper-proofing*. Teknik *tamper-proofing* harus dapat melakukan dua hal berikut:

- Mendeteksi bilamana *software* telah dimodifikasi
- Dan mencegah program dieksekusi jika modifikasi telah terdeteksi.

Pencegahan dan pendeteksian modifikasi

Ada tiga cara utama dalam pendeteksian *tampering* [1]:

- Memeriksa *executable program* itu sendiri, apakah masih identik dengan program yang asli. Untuk mempercepat proses pemeriksaan, algoritma *message-digest* seperti MD5 [13] dapat digunakan.
- Memeriksa validitas dari hasil sementara yang dihasilkan oleh program. Teknik ini dikenal sebagai *result checking*, dan telah dipuji sebagai alternatif terhadap pengujian dan verifikasi program.
- Mengenkripsi *executable*, sehingga mencegah orang lain untuk memodifikasi program kecuali ia memiliki kunci untuk mendekripsi program tersebut.

4 Kesimpulan

Dari pembahasan diatas dapat disimpulkan bahwa:

1. *Software watermarking* dapat diterapkan untuk melindungi *software* dari pembajakan oleh pihak yang tidak berhak.
2. Terdapat beragam teknik untuk menyisipkan *watermark* ke dalam *software*, baik secara statis (*static software watermarking*) maupun dinamis (*dynamic software watermarking*).
3. Tidak ada teknik *software watermarking* yang sepenuhnya tahan terhadap serangan. Pada umumnya, teknik *software watermarking* yang ada sekarang masih rentan terhadap serangan transformatif dan serangan distortif.
4. Penting bagi pengguna *software* untuk mengetahui *software* yang digunakan belum dimodifikasi oleh pihak yang tidak berhak. Oleh karena itu, terdapat beberapa teknik untuk *tamper-proofing* yang mendeteksi modifikasi pada *software* sekaligus mencegah eksekusi *software* yang telah termodifikasi.

DAFTAR PUSTAKA

[1] Collberg, C, Clark Thomborson. (2002). *Watermarking, Tamper-proofing, and Obfuscation – Tools for Software Protection*.

[2] Collberg, C, Clark Thomborson, Pradeep Kyasanur. (2002). *Watermarking, Tamper-proofing, and Obfuscation – Tools for Software Protection, Presentation Slides*.

[3] Business Software Alliance (2006). *Third Annual BSA and IDC Global Software Piracy Study, May 2006*. http://www.bsa.org/globalstudy/upload/2005%20Piracy%20Study%20-%20Official%20Version.pdf

[4] Palsberg, Jens. (2002). *Software Watermarking with Secret Keys*. Purdue University, Secure Software System Group.

[5] Ross J. Anderson and Fabien A.P. Peticolas. (1998) *On the limits of steganography. IEEE J-SAC, 16(4)*.

[6] Scott A. Moskowitz, Marc Cooperman. (1996). *Method for stega-cipher protection of computer code*. US Patent 5,745,569.

[7] Collberg, Christian, Clark Thomborson, and Douglas Low. (1998). *Breaking abstractions and unstructuring data structures*. In IEEE International Conference on Computer Languages, ICCL'98, Chicago, IL.

[8] Robert L. Davidson and Nathan Myhrvold. (1996). *Method and system for generating and auditing a signature for a computer program*. US Patent 5,559,884.

[9] Mozilla Foundation. (1995). *The Book of Mozilla*. Mozilla Foundation. http://www.mozilla.org/book/

[10] The *Easter Egg*s Archive. http://www.eeggs.com/

[11] Collberg, C, Clark Thomborson. (1998). *On The Limits of Software Watermarking*. Department of Science, The University of Auckland, New Zealand.

[12] Collberg, C, E. Carter, dkk. (2004). *Dynamic Path-Based Software Watermarking*. Department of Computer Science, University of Arizona, USA.

[13] R. Rivest. (1992). *The MD5 Message-Digest Algorithm*. The Internet Eng. Task Force RFC 1321. http://www.ietf.org/rfc/rfc1321.txt.

[14] Penjelasan Atas Undang-Undang Republik Indonesia Nomor 19 Tahun 2002 Tentang Hak Cipta. http://www.indonesia.go.id/produk_uu/isi/uu2002/uu19'02pjls.htm