ソフトウェア電子透かしに関する研究

～動的グラフ構造透かしについての検討～

2001MT044　神谷　重毅
指導教員　真野　芳久

1. はじめに

近年の電子メディア技術の発達に比べ電子メディアの著作権に関する意識が追い付いておらず、無断で複製するなどの不正利用が行われ、製作者の著作権侵害が問題になっている。

その対策として現在、ソフトウェアの著作権保護を目的としたソフトウェアプロテクションと呼ばれる技術の研究が進められており、電子透かしはその技術のひとつである。

電子透かしとは電子メディアにあらかじめ著作権情報等の情報を秘密裏に埋め込んでおき、その電子メディアが不正利用された際、著作権を主張するための技術である。

本研究では、特にプログラムに対しての透かし技術であるソフトウェア透かしに関して検討する。

2. ソフトウェア透かし

ソフトウェア透かしとは、プログラムに対して予め著作権情報などを埋め込んでおき、必要な時に埋め込んでおいた透かし情報を取り出して盗用の事実の証明とする技術である。プログラムの盗用という行為そのものを未然に防ぐことは難しい。そのため盗用の事実を立証する技術が注目されている。ソフトウェア透かしは、その埋め込みの形式から静的透かしと動的透かしの 2 つに分けて考えられる。

2.1. 静的透かし

プログラムを実行することなく、透かしを取り出すことの出来る透かしである。

2.2. 動的透かし

特定の入力に対してプログラムが実行された場合のみ、透かしが取り出せる透かしのことである。動的透かしは、ソースコードを一見しただけではそれと解らない形で透かしが埋め込まれている。入力に対するプログラムの実行があってはじめて、透かしとしての形が構築される。

3. 動的グラフ構造透かし

動的グラフ構造透かしとは、動的に構築されるグラフ構造に透かし情報を埋め込むものである。ソフトウェアに透かしを埋め込む時に、プログラムが満たすべき条件として、プログラムの仕様に影響を与えない事、攻撃を受けても透かしが消失しない事がある。

両者は相反する性質を持っている。消失しにくいように透かしを強化することは元のプログラムの表現を変えることになるからである。この問題点の解消の為、動的グラフ構造を利用した埋め込み方法の研究がされている。

動的グラフ構造として透かしを挿入する手法を取る事による利点には、複数のポインタで同一のデータを扱う事で解析が困難になる点、Javaなどのオブジェクト指向言語でのプログラムの場合ポインタ操作は自然な表現であるため透かしの判別が困難である点、プログラム中でグラフを分割して扱う事によって大量の情報を表現できる点が挙げられる

以下にその例を挙げ、それぞれのグラフ構造に対して、そのグラフによってどのような情報の表現がされているのか、1 ワードあたり何 bit の情報の埋め込みが可能であるのかを中心に検討を行う。

3.1 Radix Encoding ([3][4]より)

この方法では、埋め込みたい値をある進数に直し、循環リストに割り当てて表現される。mをリストの長さとした場合、

$0 \ldots (m+1)^m - 1$ の範囲の数値を表現できる。一般に $\log (m+1)^m / 2m = 1/2 \log (m+1)$ bit/word の埋め込みが可能であり、この値は他の例での値より大きい。

図1に基数 4 のエンコード例を示す。基本となる環状リストに加え、左側セルからのポインタで 4 進各桁の値を示している。

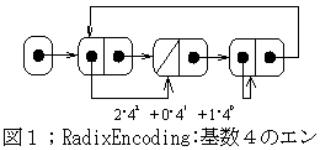

図１；RadixEncoding:基数４のエンコード

3.2 Permutation (図 2)

左のポインタで n!通りある置換のk番目の置換を表現する。埋め込むことのできる情報量は、スターリングの公式より

$\log n! / 2n \quad 1/2 \log n - O(1)$ bit/word

3.3 Parent-pointer (図 3)

ある列挙方法によって得られる木の順序数により透かしを

表現する。情報量は 1.564 — O(logn/n) bit/word

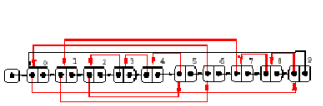

図２：Permutation([3][4]より)

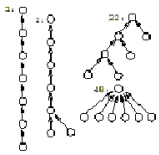

図３：Parent-Pointer([3][4]より)

ここまでに述べたものはエラー修正の特性が無く、ノードの追加等の攻撃を受けた場合に本来のものとは別の透かし情報として認知される。

4. 透かしの強化･拡張

プログラムに透かしを埋め込む際に動的構造の手法を用いることで、ある程度の攻撃の耐性は得られるが、それだけで万全ではない。動的構造の透かしに対する攻撃手法として考えられるものに、Split（分節攻撃）、ダミーポインタフィールドの追加、根の発見の妨害、フィールド名や順序の変更等がある。これらの攻撃に耐性を持たせるため、グラフ構造の強化･拡張をする必要がある。その例を挙げる。

4.1 グラフ構造の強化

図 4 はグラフ構造の節点を 2 つに分割する攻撃に対してのグラフ構造の強化を表したものである。攻撃はグラフの節点を分断することで、本来のグラフ構造の形を壊し透かしを消滅あるいは無意味な情報に変更することを想定している。攻撃はグラフの節点を分断することで、本来のグラフ構造の形を壊すし透かしを消滅あるいは無意味な情報に変更することを想定している。各ノード部分を 3 つのノードの集合とし、それぞれのノードにはそれまで1つのノードで扱っていたポインタを分割して割り当てる。このことで集合の一部が分断されても、それぞれの節は残るためにグラフ構造としての全体の形は守られる。

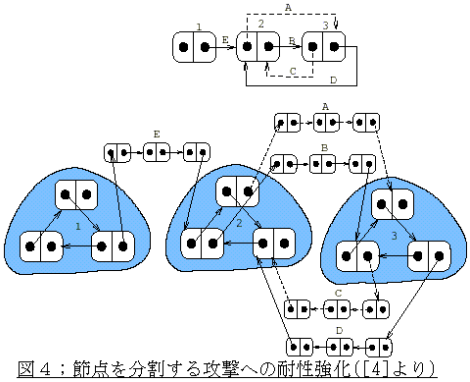

図４：節点を分割する攻撃への耐性強化([4]より)

4.2 Radix 手法の攻撃耐性強化

先に紹介した Radix 手法の強化について検討する。Radix 手法によるリスト構造が節点を2つに分割する攻撃を受けた場合、基本の循環リストとしての形は残るが埋め込まれている透かしはリスト構造の形そのものである為、秘密情報としての機能が果たせなくなる（図5）。

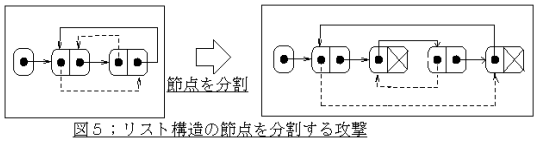

図５：リスト構造の節点を分割する攻撃

ここで Radix 手法に 5.1 で紹介した攻撃への耐性の手法を加えたグラフ構造を検討する。各ノード部分を 3 つのノードの集合とすることで節点を 2 つに分割する攻撃を受けても元のリストを復元することが出来、攻撃に対しての耐性を持つことが出来る。これにより、この複合グラフ構造はダミーポインタフィールドの追加と節点の分割、2 つの攻撃に耐性を持った新しいグラフ構造とすることが出来る。

4.3 拡張

先に紹介した Radix Encoding の手法は、1 語あたりの情報量を大きく出来るという点で動的グラフとして効率の良いものだった。さらに効率性を良くする為の拡張例を検討する。リストのセル部分にさらに1つのポインタを追加する場合を考える。この場合、メモリは 1.5 倍になるが、実際に表現することのできる透かしの情報量は 2 倍となる（図6）。

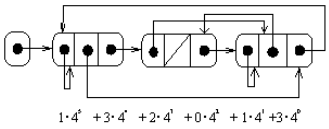

図６：リストの拡張

5. おわりに

グラフ構造透かしの例を示し、分割による攻撃への耐性の強化について検討した。拡張方法もさらに工夫することで情報量の増大が見込める。独自のグラフ構造の創造には至らなかったが、数学的な面･プログラムの仕様の面等、様々な視点から考えることで新しいグラフ構造透かしの可能性が見えてくる。

参考文献

[1] C.Collberg, C.Thomborson: Watermarking,Tamper-Proofing, and Obfufscation – Tools for Software Protection, IEEE Transactions on SE, Vol.28 No.8 (2002.8).

[2] C.Collberg, C.Thomborson: Software Watermarking Models and Dynamic Embeddings, ACM POPL (1999).

[3] C.Collberg, C.Thomborson: Error-Correcting Graphs for Software Watermarking, (WG 2003), July 2003.

[4] C.Collberg, C.Thomborson: Dynamic Graph-Based Software Watermarking, Univ of Arizona TR04-08, April 2004.