コンピュータセキュリティ 10－7
(2000. 7. 25)

プログラムの冗長化に関する検討

中村 直己[†], 西垣 正勝[††], 曽我 正和[†††], 田窪 昭夫[††††]

[†] 静岡大学大学院情報学研究科 〒432-8011 浜松市城北 3-5-1

[††] 静岡大学情報学部情報科学科 〒432-8011 浜松市城北 3-5-1

[†††] 岩手県立大学ソフトウェア情報学部 〒020-0173 岩手県滝沢村滝沢字巣子 152-52

[††††] 三菱電機情報システム製作所 〒247-8520 鎌倉市上町屋 325

E-mail: nisigaki@cs.inf.shizuoka.ac.jp

あらまし：
プログラムの一部に冗長行を加えてもプログラムの実行結果に変わりはない。ここで、プログラムのどこにどのような冗長行を加えたかということを一種の「情報」として扱うことが可能である。つまり、プログラムの冗長化はプログラムそのものに情報を埋め込むことに値すると言える。この冗長化情報を秘密に保つことにより、プログラムの冗長化をセキュリティ技術の一つとして用いることができる。本稿では冗長化プログラムによる電子透かし、ステガノグラフィ、認証を例にとり、プログラムの冗長化のセキュリティ技術としての可用性を議論する。

キーワード：冗長性、プログラム、セキュリティ、電子透かし、ステガノグラフィ、認証

A discussion on the redundancy of the programs

Naoki NAKAMURA[†], Masakatsu NISHIGAKI[††], Masakazu SOGA[†††] and Akio TAKUBO[††††]

[†] Graduate school of Information, Shizuoka University,

[††] Department of Computer Science, Faculty of Information, Shizuoka University,
3-5-1 Johoku, Hamamatsu, 432-8011, Japan

[†††] Faculty of Software and Information, Iwate Prefectural University,
Sugo 152-52, Takizawa, Iwate, 020-0173, Japan

[††††] Mitsubishi Electric Corp, 325 Kamimachiya, Kamakura, 247-8520, Japan
E-mail: nisigaki@cs.inf.shizuoka.ac.jp

Abstract:
This paper describes the security systems based on the redundancy on the programs. Supposing that the locations of the redundancy are kept secret, it would take too much time for any adversary to reveal where and how the is added to the programs.Therefore, it is possible to embed the secret information into the programs, and it is expected to develop the security systems using the redundant programs.
Here, we show three application examples, such as watermark, steganography and authentication.

Keywords : redundancy, program, security, watermark, steganography, authentication

1 はじめに

近年のIT技術の向上により、もはやコンピュータは生活・ビジネス・コミュニケーションのツールの一つとしてなくてはならないものとなった。しかしこれにともない、コンピュータ犯罪もまた増加、高度化の一途をたどっている。また、クラッキングの技術やツールがインターネット上に氾濫し、専門知識のないユーザでさえも犯罪行為を行うことができるようになるなど、新たな問題も発生している。

この現状に対し、様々なセキュリティ技術やデバイスが研究されている。しかし犯罪技術も進歩しており、特に計算機の高速化により総当り攻撃の実用性が高まっている。また、インターネット人口の増加にともない、セキュリティ意識の低いユーザの絶対数が増えていることも事実である。これらを総じて考えると、セキュリティ強度は下がる一方であると言えよう。今後、セキュリティの重要性は益々大きくなる。

さて、ここで「セキュリティ」の役割を考えてみる。一般的に、セキュリティに関する処理はアプリケーションプログラムに行わせたい仕事の本質ではない。理想的な環境(善人のみのネットワーク)においてはセキュリティは不要である。例えばユーザ認証(識別)を行いたいのならば、IDのみのチェックで事足りる。すなわち、アプリケーションプログラムが行う仕事から見れば、セキュリティチェックは冗長な機能であると言える。逆に言えば、アプリケーションプログラムに冗長な機能を付加することによりセキュリティを強化しているのだととらえることができる。

特に文献[1]-[5]などは「冗長性」を積極的に利用したセキュリティ技術であるといえる。文献[1][2]では、プログラムの中に条件が必ずfalseとなるif文やwhile文などを用意し、実行されない(冗長な)命令のオペランドなどに秘密情報を隠し、これを電子透かしとする事が提案されている。また、文献[3]は、Javaプログラムに挿入された電子透かしの改竄を防ぐために、プログラムを難読化(複雑に冗長化)している。一方、文献[4][5]では、データベースやWebページの二重化という冗長化によってデータの正当性を保証しようと試みている。

本稿では、プログラムのどこにどのような冗長行を加えたかということを一種の「情報」として扱う。この冗長化情報を秘密に保つことにより、セキュリティを維持する方式を提案する。例として冗長化プログラムによる電子透かし、ステガノグラフィ、認証を採りあげ、本方式の可用性を検討する。

2 冗長化プログラム

2.1 プログラムの冗長性

プログラムの一部にいくつかの命令を加えてもプログラムの実行結果に変わりがない場合、加えられた命令群を「冗長命令」と呼ぶことにする。なお、プログラムにおいては一行が一命令となる慣習があるため、場合に応じて冗長命令を「冗長行」とも呼ぶ。また、オリジナルプログラムに冗長命令が加わったものを「冗長化プログラム」と呼ぶことにする。

図1にプログラムの冗長化の例を示す。図1(b)の16,17行目が挿入された冗長命令である。また、図1(c)の15〜17行目のように、複数命令によってある命令を実現することも冗長化の一方式と考える。これら三つのプログラムの実行結果はすべて等しい。

なお、ここではC言語のソースファイルを例に説明したが、基本的にはいかなる形式のプログラムも同様の方法で冗長化することが可能である。ただし、ソースプログラムを対象に冗長化を考える場合には、コンパイラの最適化により挿入された冗長命令が削除されないような工夫を施す必要がある。本稿では、説明の簡単化のために以後すべての説明においてもC言語のソースファイルを用いることとする。

2.2 冗長挿入法

冗長命令挿入の条件としては、冗長化プログラムがオリジナルプログラムと同じ実行結果となるということさえ満足すれば良い。したがって2.1節の例

```
14 : ・・・
15 : x=(x+y)/2;
16 : printf("%d",x);
17 : ・・・
```
(a)オリジナルプログラム

```
14 : ・・・
15 : x=(x+y)/2;
16 : x=x+y;
17 : x=x-y;
18 : printf("%d",x);
19 : ・・・
```
(b)冗長化プログラム1

```
14 : ・・・
15 : x=2*x;
16 : x=x+2*y;
17 : x=x/4;
18 : printf("%d",x);
19 : ・・・
```
(c)冗長化プログラム2

図 1: プログラムの冗長例 1

```
14 : ・・・
15 : x=(y+z)/2;
16 : printf("%d",x);
17 : ・・・
```
(a)オリジナルプログラム

```
14 : ・・・
15 : x=(y+z)/2;
16 : y=x AND y;
17 : x=x OR y;
18 : y=2*x-z;
19 : printf("%d",x);
20 : ・・・
```
(b)論理演算を利用した冗長化プログラム

```
14 : ・・・
15 : x=(y+z)/2;
16 : x=4*x;
17 : x=x >>2;
18 : printf("%d",x);
19 : ・・・
```
(c)シフト演算を利用した冗長化プログラム

図 2: プログラムの冗長例 2

に限らず

- 変数やパラメータに、最終的には値が元に戻るように各種演算を施す
- オリジナルプログラムにおいては必要でない変数やパラメータ、関数を定義して使用する
- 必ず false となる条件分岐を含んだ if 文や while 文などを付加する

などの方法により冗長命令を挿入する事ができる。もちろんこれらを組み合わせて使用しても良い。基本的には、オリジナルプログラムのいかなる場所にも冗長命令を挿入することが可能である。

なお、本稿はプログラムの冗長化という概念を示すものであり、特定の冗長命令挿入法を論ずるものではないことを明記しておく。

2.3 冗長性とセキュリティ

プログラムのどこにどのような冗長性が加えられたかということが一種の「情報」となる。すなわち、プログラムの冗長化はプログラムそのものに情報を埋め込むことであるととらえることができる。この冗長化情報を秘密に保つことができれば、プログラムの冗長化をセキュリティ技術の一つとして用いることが可能であると推測される。

プログラムの意味がわからなければ、どこが冗長であるか判断できない。したがって、一般的に冗長化プログラムから冗長命令を見つけるためには全文解析が必要となる。図 1 の例では変数 x に関するプログラムスライシング [6] により冗長化プログラムの 15〜18 行目のみが抜き出されてしまうと、冗長性の解析は容易となる。しかし基本的にどの変数や関数が冗長化の対象になっているかを特定することは難しいので、結局攻撃者はすべての変数に関するプログラムスライスを生成して冗長性の有無を調べなくてはならない。これは最悪、プログラムの全文解析に匹敵する負荷となる。

また、図 2 の例のような工夫により冗長命令の発見はより困難になるものと思われる。ここで、図 2(b) では論理演算の恒等式 $x{=}{=}x \vee x \wedge y$ を利用しており、

図 2(c) では 2^n の倍数である整数 x の n ビット論理右シフトが x の $1/n$ 倍に等しいということを利用している。(ただし、この例では 16 行目の命令 x=4*x; によって x がオーバフローしないという保証が必要である。)

一方、オリジナルプログラムと冗長化プログラムの両方を有する者は、両プログラムを比較 (パターンマッチング) することによって、容易に冗長命令の存在箇所を特定することができてしまう。したがって、冗長命令に関する情報を秘密に保ちたい場合には、オリジナルプログラムを配布してはならない。同様に、1 つのオリジナルプログラムから複数種の冗長化プログラムを作ることも避けるべきである。複数種の冗長化プログラムを異なるユーザに配った場合も、ユーザが結託して冗長化プログラムどうしを比較することにより、(オリジナルプログラムがなくとも) 冗長命令の存在箇所が発見されてしまう。

3 冗長化プログラムによる情報ハイディング

2 章でプログラムの冗長化がプログラムに情報を埋め込むことと等価であることを示した。その情報を秘密裡に保持することにより、冗長化プログラムによる情報ハイディングを実現することができる。

3.1 電子透かし

プログラムのどこにどのような冗長命令を挿入したかという情報を著作権情報として用いることにより、プログラムに電子透かしを埋め込むことができる。手順を以下に示す。

1. プログラムの著作者はオリジナルプログラムを作成する。
2. 著作者はオリジナルプログラムに適当に冗長命令を挿入し、プログラムを冗長化する。
3. 著作者はオリジナルプログラムと冗長命令をどこに挿入したかという情報を公の秘密情報保管機関に登録する。
4. 著作者はプログラムの購入者に対して冗長化プログラムを配布する。
5. 著作権の侵害が疑われるプログラムが見つかった場合には、著作者 (または公の秘密情報保管機関) がそのプログラムの冗長命令を調べることにより、プログラムの身元を検証することができる。

ここで、2.3 節で述べたように、一つのオリジナルプログラムから複数種の冗長化プログラムを作ってはいけない。したがって、電子透かしとして埋め込むことのできる情報は「このプログラムは○○が作成した」という情報だけである。購入者ごとに個別に「このプログラムは○○が作成し××に売った」という透かし情報を埋め込むと、購入者の結託により冗長命令の挿入箇所が発見されてしまう。そこで、著作者はプログラムを誰に売ったのかという情報については別に保管しておく必要がある。図 3 にこれを考慮した一連の手順を図示した。

複数箇所に冗長命令を挿入しておいたほうがチェックにおける確実性が高まる。複数の冗長性があれば、冗長化プログラムが解析された場合にも、すべての冗長命令を見つけられない限り透かしの効力が残る。また、プログラムの著作権をサブルーチン単位で守りたい場合には、当然サブルーチンやモジュールごとに冗長命令を挿入していかなければならない。ただし、冗長命令が挿入されるにしたがいプログラムの実行速度は落ちるので、冗長命令を過度に挿入し過ぎることは避けるべきであろう。

3.2 ステガノグラフィ

冗長命令を挿入する方法を規則化し、二者が前もってその規則を秘密裡に共有しておくことにより、冗長化プログラムをステガノグラフィに利用することができる。A が B に秘密メッセージを含んだ冗長化プログラムを送る手順を以下に示し、図 4 にこれを図示する。

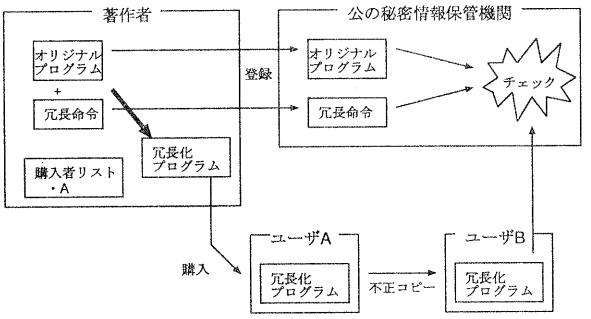


図 3: 電子透かしシステム

1. A と B が前もって冗長命令を挿入する規則を作成し、秘密裡に共有しておく。
2. A は秘密メッセージ m とあるアプリケーションプログラム p を用意する。
3. A は p をオリジナルプログラムとし、冗長命令を挿入する規則にしたがい、m に相当する冗長命令を p に付加し、冗長化プログラム p' を作成する。
4. A はオリジナルプログラム p を消去する。
5. A は冗長化プログラム p' を B に送信する。
6. B は冗長命令を挿入する規則に従って、冗長化プログラム p' から m を取り出す。

ここで手順 5 は、オリジナルプログラムが盗まれ、冗長化プログラムと比較されることにより秘密メッセージが推測されることを防止するために行う。また、B が A に秘密メッセージを送る場合も同様の手順となる。

第三者 C から見ると当事者 A と B がいろいろなプログラムを物々交換しているように見える。オリジナルプログラムと冗長化プログラムは全く同じ動作結果を呈するので、A と B が送受信しているプログラムを C が手に入れて実行しても不信な点は見当たらない。プログラム中に冗長性が存在し、かつ、その冗長性に意味があるということに気付かない限り、C は A-B 間の秘密通信を認識できない。

ただし、アプリケーションプログラムの機能に比べてファイルサイズが大き過ぎると C に怪しまれる。よって、プログラムの機能に見合う量の冗長性の付加が許されることになる。逆に言えば、大規模なアプリケーションプログラムをオリジナルプログラムとして選んだ場合、冗長命令をある程度多く挿入しても不自然には見えない。本方式によれば、大規模なアプリケーションプログラムを用いて、一度に (一つのプログラムによって) 大量の秘密情報を発信することが可能である。一方、本方式においても、一つのオリジナルプログラムから複数種の冗長化プログラムを作ってはいけないという 2.3 節の制約が課せられる。したがって、秘密メッセージごとにオリジナルプログラムを用意する必要がある。このため、双方向の秘密通信を頻繁に行いたい場合には本方式は不向きであろう。

4 冗長化プログラムによる認証

2.3 節で考察したように、一般的に冗長化プログラムから冗長命令を見つけることは難しい。逆に言えば、冗長化プログラム中の冗長命令を正しく示すことができるのは、冗長命令の存在箇所を前もって知っているユーザのみである。これを利用することにより、プログラムの冗長性を基盤とした認証方式を構築することができる。

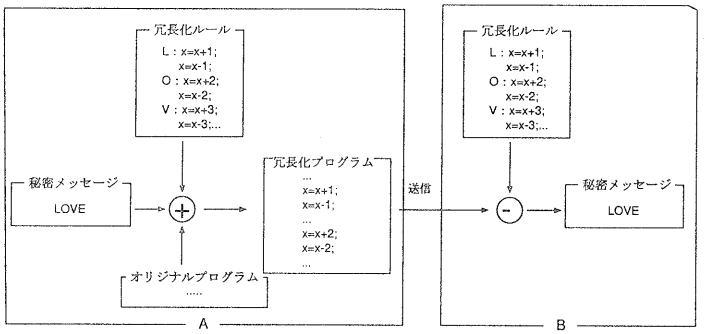


図 4: ステガノグラフィシステム

4.1 冗長命令の削除による認証

プログラムの冗長性を基盤とした認証方式の手順は以下のようになる。また図 5 にこれを図示する。

1. サーバ S は t を入力としてある関数値 $f(t)$ を計算するプログラムを用意し、これをオリジナルプログラム p_A とする。
2. S はオリジナルプログラム p_A に冗長命令を加え、冗長化プログラム p'_A を作る。ここで、どこにどのような冗長命令を加えたかが秘密情報 k_A となる。
3. S はユーザ U_A を登録し、U_A に秘密情報 k_A を (安全な方法により) 与える。
4. U_A は S にログインする際に k_A を掲示する。
5. S は掲示された k_A に記されている冗長命令を p'_A から削除することにより、冗長化プログラム p'_A をプログラム p''_A に変更する。
6. S は乱数 r を発生する。
7. S は r を冗長化プログラム p'_A に入力することにより $f(r)'$ を計算する。(r をオリジナルプログラム p_A に入力することにより $f(r)$ を計算しても等価である)
8. S は r を手順 5 で作られたプログラム p''_A に入力することにより $f(r)''$ を計算する
9. $f(r)'=f(r)''$ が成立する場合に限り、S は U_A からの接続要求であることを認め、ログインを許す。
10. 手順 4 に戻る。

4.2 複数ユーザの認証

4.1 節の認証方式ではサーバは各ユーザ U_i ごとに冗長化プログラム p'_i を用意し、それを保管しなければならなかった。しかし、一つの冗長化プログラム p' のみを用いて、複数ユーザの認証を行うことも可能である。

図 6(a),(b) に二人のユーザ A および B を認証する際にサーバが用意するオリジナルプログラム p と冗長化プログラム p' の一例を示した。冗長化プログラム p' の 16〜21 行目が冗長命令である。ここで、冗長化プログラム p' から 16〜18 行目を削除したプログラムも 19〜21 行目を削除したプログラムもオリジナルプログラム p と同じ実行結果を出力するように冗長命令が挿入されている事が重要である。サーバは p' 中の 16〜18 行をユーザ A 用の秘密情報 k_A(図 6(c)) とし、p' 中の 19〜21 行をユーザ B 用の秘密情報 k_B(図 6(d)) として各ユーザに与える。この結果、一つの冗長化プログラム p' にて、ユーザ A,B の両方を 4.1 節の認証方式に準ずる手順により認証することが可能となる。

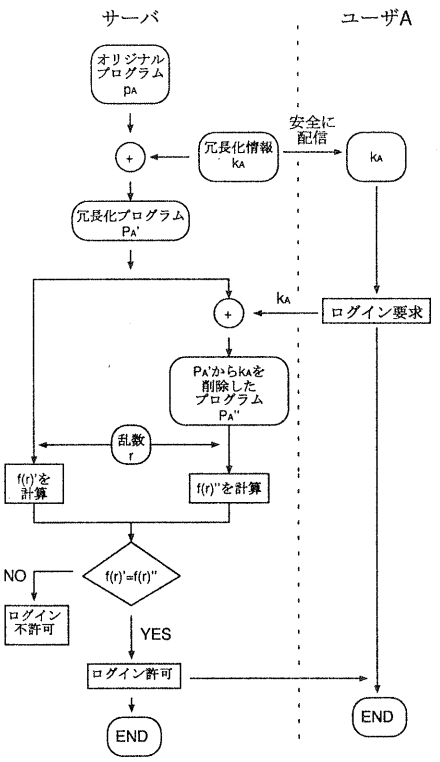


図 5: 認証方式

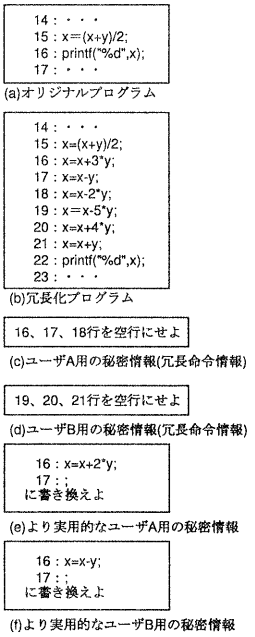


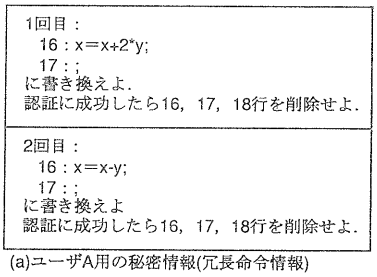
図 6: 複数ユーザ認証のためのプログラムの冗長化の例

ただし、本方式は実用的には問題がある。図 6(b) の冗長化プログラムは、16〜18 行または 19〜21 行を削除した場合のみでなく、15〜21 行、17 行と 21 行、16 行と 18 行〜20 行を削除した場合にも処理結果が等価となってしまう。すなわち、この方式では予期しない秘密情報が発生する可能性がある。これに対処するためには、図 6(e),(f) のようにユーザ A とユーザ B の秘密情報を「どの行を消すのか」という情報から「どの行をどのように書き換えるのか」という情報に拡張して、セキュリティ強度を向上させるという手段が考えられる。

4.3 ワンタイムパスワード型認証方式

4.2 節の認証方式を一人のユーザを認証するための方式として用いることにより、ワンタイムパスワード型の認証を行うことも可能である。この場合、図 6(a),(b) の例に対して、図 7(a) がユーザに与えられる秘密情報 (冗長行情報) となる。

本方式は基本的に 4.2 節の認証方式と同じである。しかし認証が成功した場合には冗長化プログラムから当該冗長命令が削除されるという点が異なる。例えば、1 回目の認証が成功した後では、図 6(b) であった冗長化プログラムは図 7(b) のように変更されている。冗長化プログラムが図 7(b) となった時点で図 7(a) に記されている 1 回目の認証用の秘密情報は無効となる。かつ、この時点で初めて図 7(a) に記されている二回目の認証用の秘密情報が有効となる。



(a)ユーザA用の秘密情報(冗長命令情報)

```
14：・・・
15：x=(x+y)/2;
16：x=x-5*y;
17：x=x+4*y;
18：x=x+y;
19：printf("%d",x);
20：・・・
```

(b)1回目の認証が成功した後の冗長化プログラム

図 7: ワンタイムパスワード型認証のための冗長命令情報とプログラム変化

なお、本方式においては各ユーザ U_i ごとに冗長化プログラム p_i' を用意する必要がある。

5 まとめ

プログラムの冗長性をセキュリティ技術に応用する方法を提案した。本稿では冗長化プログラムによる電子透かし、ステガノグラフィ、認証を例にとり、プログラムの冗長化のセキュリティ技術としての可用性を議論した。

今後は各方式の詳細設計に移るとともに、それらのセキュリティ強度を評価することが大きな課題となる。また、本方式の実用化においては、バイナリプログラムを効率的に冗長化する方法を確立する必要がある。しかしプログラムを冗長化する方法は本稿で紹介したものに留まらない。新たなプログラムの冗長化法が見つかれば、それに適した別のセキュリティ技術への適用も可能となってくると思われる。

謝辞

本研究は (財) テレコム先端技術センターおよび (株)NTT データの助成を受けた。ここに謝意を表する。

参考文献

[1] 門田, 飯田 : "プログラムに電子透かしを挿入する一手法", 暗号と情報セキュリティシンポジウム論文集, SCIS98-9.2.A, 1998.

[2] 北川, 楫, 関 : "難読化後も検出可能な JAVA プログラムに対する電子透かし法", コンピュータセキュリティシンポジウム'98(CSS'98) 論文集, pp.63-68, 1998.

[3] 北川, 楫, 嵩 : "JAVA で記述されたプログラムに対する電子透かし法", 暗号と情報セキュリティシンポジウム論文集,SCIS98-9.2.D, 1998.

[4] J.McDermott, R.Gelinas, S.Ornstein : "Doc, Wyatt, and Virgil: Prototyping Storage Jamming Defenses", Proc. COMPUTER SECURITY APPLICATIONS Conference , pp.265-273, 1997.

[5] 可部, 西垣, 曽我, 田窪, : "ホームページの改竄パトロール方式", 情報処理学会研究報告, 2000-CSEC-8-30, pp.173-178, 2000.

[6] F.Tip, "A survey of program slicing techniques", J. of programming languages, vol.3, pp.121-189, 1995.