

(19) 日本国特許庁(JP)

(12) 特許公報(B2)

(11) 特許番号

特許第3944069号
(P3944069)

(45) 発行日 平成19年7月11日(2007.7.11)

(24) 登録日 平成19年4月13日(2007.4.13)

(51) Int. Cl. F I
G06F 21/22 (2006.01) G06F 9/06 660L

請求項の数 11 (全 19 頁)

(21) 出願番号	特願2002-355881 (P2002-355881)	(73) 特許権者	504143441
(22) 出願日	平成14年12月6日(2002.12.6)		国立大学法人 奈良先端科学技術大学院大 学
(65) 公開番号	特開2004-192068 (P2004-192068A)		奈良県生駒市高山町8916-5
(43) 公開日	平成16年7月8日(2004.7.8)	(74) 代理人	100067828
審査請求日	平成16年4月2日(2004.4.2)		弁理士 小谷 悦司
		(74) 代理人	100096150
			弁理士 伊藤 孝夫
		(74) 代理人	100099955
			弁理士 樋口 次郎
		(74) 代理人	100109438
			弁理士 大月 伸介
		(72) 発明者	門田 暁人
			奈良県生駒市高山町8916-5 奈良先 端科学技術大学院大学内
			最終頁に続く

(54) 【発明の名称】 自己書換え処理追加プログラム、自己書換え処理追加装置及び自己書換え処理追加方法

(57) 【特許請求の範囲】

【請求項1】

保護対象プログラムを構成する複数の命令コードのうちいずれか1カ所の命令コードを決定する命令コード決定手段、

決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装する命令コード偽装手段、

前記偽の命令コードを元の正しい命令コードに戻す第1のルーチンと当該第1のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度偽装する第2のルーチンとを生成するルーチン生成手段、

前記第1及び第2のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成する自己書換えプログラム生成手段としてコンピュータを機能させ、

前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第2のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第2のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第1のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第1のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数

10

20

の行の中からランダムに1つの行を前記第2のルーチンの追加位置として決定することを特徴とする自己書換え処理追加プログラム。

【請求項2】

前記命令コード決定手段が前記命令コードを決定し、次いで前記命令コード偽装手段が前記命令コードに対応する偽の命令コードを決定し、次いで前記ルーチン生成手段が前記命令コードに対応する前記第1及び第2のルーチンを生成する処理を、複数回繰り返すことにより、複数の命令コードを決定し、前記複数の命令コードのそれぞれに対応する偽の命令コードを決定し、前記複数の命令コードのそれぞれに対応する前記第1及び第2のルーチンを生成することを特徴とする請求項1記載の自己書換え処理追加プログラム。

【請求項3】

前記命令コード偽装手段は、前記正しい命令コードの機械語表現サイズと同一サイズの機械語表現を有する命令コードを前記偽の命令コードとして決定することを特徴とする請求項1又は2記載の自己書換え処理追加プログラム。

【請求項4】

前記偽の命令コードは、前記正しい命令コードの機械語表現のコード列のうち1バイトを置き換えた機械語表現を有する命令コードであることを特徴とする請求項3記載の自己書換え処理追加プログラム。

【請求項5】

前記第1のルーチンは、前記正しい命令コードの機械語表現のコード列と前記偽の命令コードの機械語表現のコード列との相違する箇所のみを前記偽の命令コードに対して書き込み、

前記第2のルーチンは、前記正しい命令コードの機械語表現のコード列と前記偽の命令コードの機械語表現のコード列との相違する箇所のみを前記正しい命令コードに対して書き込むことを特徴とする請求項1～4のいずれか1項に記載の自己書換え処理追加プログラム。

【請求項6】

前記保護対象プログラムはアセンブリプログラムであり、

前記自己書換えプログラムをアセンブルして機械語プログラムを得るアセンブル手段をさらに備えることを特徴とする請求項1～5のいずれか1項に記載の自己書換え処理追加プログラム。

【請求項7】

機械語プログラムを逆アセンブルする逆アセンブル手段をさらに備えることを特徴とする請求項6記載の自己書換え処理追加プログラム。

【請求項8】

前記アセンブリプログラムは、所定の高級言語で記述されたソースプログラムをコンパイルして得られたものであり、

前記ソースプログラムをアセンブリプログラムにコンパイルして前記アセンブリプログラムを得るコンパイル手段をさらに備えることを特徴とする請求項6又は7記載の自己書換え処理追加プログラム。

【請求項9】

機械語プログラムを連係編集するリンク手段をさらに備えることを特徴とする請求項8記載の自己書換え処理追加プログラム。

【請求項10】

保護対象プログラムを構成する複数の命令コードのうちいずれか1カ所の命令コードを決定する命令コード決定手段と、

決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装する命令コード偽装手段と、

前記偽の命令コードを元の正しい命令コードに戻す第1のルーチンと当該第1のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度

10

20

30

40

50

偽装する第 2 のルーチンとを生成するルーチン生成手段と、

前記第 1 及び第 2 のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成する自己書換えプログラム生成手段とを備え、

前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第 2 のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第 2 のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに 1 つの行を前記第 1 のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第 1 のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに 1 つの行を前記第 2 のルーチンの追加位置として決定することを特徴とする自己書換え処理追加装置。

10

【請求項 1 1】

命令コード決定手段が、保護対象プログラムを構成する複数の命令コードのうちいずれか 1 カ所の命令コードを決定するステップ、

命令コード偽装手段が、決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装するステップ、

ルーチン生成手段が、前記偽の命令コードを元の正しい命令コードに戻す第 1 のルーチンと当該第 1 のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度偽装する第 2 のルーチンとを生成するステップ、

20

自己書換えプログラム生成手段が、前記第 1 及び第 2 のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成するステップから構成され、

前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第 2 のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第 2 のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに 1 つの行を前記第 1 のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第 1 のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに 1 つの行を前記第 2 のルーチンの追加位置として決定することを特徴とする自己書換え処理追加方法。

30

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、解析及び改ざんが困難なプログラムを作成する技術に関する。

【0002】

【従来の技術】

従来より、解析及び改ざんの困難なプログラムを作成する技術がいくつか提案されており、それらの技術は「プログラムの難読化」、「プログラムの暗号化」及び「プログラムの断片化」の 3 つに大別される。いずれの方法もプログラムの解析及び改ざんに要するコスト（労力、時間）を増大させる効果を有している。

40

【0003】

図 9 は、典型的なプログラムの難読化の仕組みを示した図である。プログラムの難読化は、与えられたプログラムを複雑化して読みにくいプログラムに変換することである。開発された元のプログラム A 1 は、難読化装置 A 2 によって所定の難読化処理が施され、難読化されたプログラム A 3 として出力される。難読化されたプログラム A 3 は、元のプログラム A 1 の表現や計算手順を複雑化することにより得られ、人間にとって解析が困難となっている。そのため、難読化されたプログラム A 3 が配布された使用者（ユーザ）や第三

50

者は、プログラムを解析することが困難となり、プログラムが解析される危険性を減らすことができる。この難読化されたプログラム A 3 は、元のプログラム A 1 と同様、計算機上でそのまま実行することができる。

【 0 0 0 4 】

プログラムの難読化の具体的な方式としては、プログラムの制御構造を複雑にする方式（非特許文献 1～3）、複雑な処理を行う命令コードを複数の単純な処理を行う命令コードに置き換える方式（非特許文献 4、5）、実行結果に影響を与えない命令コードを挿入する方式（非特許文献 6、7）、データ構造を変形する方式（非特許文献 8）、プログラム中の変数名や手続き名を変換する方式（特許文献 1）、配列やポインタの参照、代入を利用してプログラムを複雑化する方式（非特許文献 9、10）などが存在する。

10

【 0 0 0 5 】

図 10 は、典型的なプログラムの暗号化の仕組みを示した図である。プログラムの暗号化は、プログラム全体又は一部を暗号化することによって、解析を困難にする技術である。開発された元のプログラム B 1 は、暗号化装置 B 2 によって、その全体またはその一部に対して所定の暗号化処理が施され、暗号化されたプログラム B 3 として出力される。暗号化されたプログラム B 3 は、暗号化された命令コード B 3 1 及び暗号化された命令コード B 3 1 を復号するためのモジュール B 3 2 から構成されている。

【 0 0 0 6 】

暗号化された命令コード B 3 1 は、人間が読んでもその内容を理解することができない。また、暗号化された命令コード B 3 1 は、そのままの状態では、計算機によって実行することができない。そのため、プログラムの実行前、又は実行中に必ず復号（暗号の逆変換）しなければならない。したがって、暗号化された命令コード B 3 1 を復号するためのモジュール B 3 2 が必要となる。このモジュール B 3 2 は、暗号化されたプログラム B 3 に追加する、又はハードウェアにより復号処理を行う場合は、予めハードウェアにモジュール B 3 2 を追加しておく必要がある。プログラムの暗号化の具体的な方式は、非特許文献 11、12、特許文献 2～5 により提案されている。

20

【 0 0 0 7 】

図 11 は、典型的なプログラムの断片化の仕組みを示した図である。プログラムの断片化は、図 11 に示すように、開発された元のプログラム C 1 は、断片化装置 C 2 によって、所定の処理が施され、断片化されたプログラム C 3 として出力される。断片化されたプログラム C 3 は、分割された多数のプログラム断片 C 3 1 及びそれらの実行を制御するモジュール C 3 2 から構成されている。人間は、個々プログラム断片 C 3 1 自体を理解することができるが、これらは断片化されているため、プログラム全体の動作を理解することができない。そのため、プログラムの解析が困難となる。なお、プログラムの断片化は、上記プログラムの難読化又は暗号化と併用される場合もある。断片化の具体的な方式は、非特許文献 13、14、特許文献 6～8 などにおいて提案されている。

30

【 0 0 0 8 】

【非特許文献 1】

門田暁人、高田義弘、鳥居宏次、"プログラムの難読化の提案"、情報処理学会第 51 回全国大会講演論文集、5G-7、pp.4-263-4-264、1995。

40

【非特許文献 2】

門田暁人、高田義弘、鳥居宏次、"ループを含むプログラムを難読化する方法の提案"、電子情報通信学会論文誌、D-1、Vol.J80-D-1、No7、pp644-652、July 1997。

【非特許文献 3】

FritzHohl,"Time limited blackbox security:Protecting mobile agents from malicious hosts",In G.Vigna ed.Mobile Agents Security,Lecture Notes in Computer Science, Vol1419,pp.92-113,Springer-Verlag,1998.

【非特許文献 4】

村山隆徳、満保雅浩、岡本栄司、植松友彦、"ソフトウェアの難読化について"、電子情報通信学会技術研究報告、ISEC95-25、Nov.1995。

50

【非特許文献 5】

M.mambo,T.Murayama,and E.Okamoto,"A tentative approach to constructing tampei-resistant software",In Proc.New Security Paradigm Workshop,cumbia,UK,1997.

【非特許文献 6】

C.Collberg,C.Thomborson,and D Low,"A taxonomy of obfuscating transformations",Technical Report of Dept.of Computer Science,U.of Auckland,No.148,NewZealand,1997.

【非特許文献 7】

C.Collberg,C.Thomborson,and D.Low,"Manufacturing Cheap,Resilient,and Stealthy Opaque Constructs",ACM AIGPLAN-SIGACT Symposium of Dept.of Computer Languages(POPL 98),San Diego,California,1998.

10

【非特許文献 8】

C.Collberg,C.Thomborson,and D Low,"Breaking Abstractons and Unstructuring Data Structures",IEEE International Conference on Computer Languages(ICCL'98),Chicago,IL,May 1998

【非特許文献 9】

C.Wang,J.Hill,J.Knight,and J.Davidson,"Software tamper resistance :Obfuscating static analysis of programs",Technical Report SC-2000-12,Department of Computer Science,University of Virginia,Dec.2000

【非特許文献 10】

Toshio Ogiso,Yusuke Sakabe,Masakazu Soshi,and Atsuko Miyaji,"Software tamper resistance based on the difficulty of interprocedural analysis",In Proc.International Workshop on Information Security Application(WISA2002),Lecture Notes in Computer Science,Springer-Verlag,2002

20

【非特許文献 11】

D.J.Albert and S.P.Morse,"Combating software piracy by encryption and key manege ment",Computer,April 1982

【非特許文献 12】

石間宏之、齋藤和夫、加盟光久、申吉浩、"ソフトウェアの耐タンパー化技術"、富士ゼロックステクニカルレポート No.13,pp.20-28,2000.

【非特許文献 13】

David Aucsmith,"Tamper Resistant Software:An Implementation",In R.J.Anderson ed. Information Hiding Workshop,Lecture Notes in Computer Science,Vol.1174,pp.317-333,1996

30

【非特許文献 14】

鴨志田昭輝、松本勉、井上信吾、"耐タンパーソフトウェアの構成手法に関する考察"、信学技報、ISEC97-59,pp.69-78,1997.

【特許文献 1】

United States Patent 6,102,966

【特許文献 2】

United States Patent 4,278,837

40

【特許文献 3】

United States Patent 4,847,902

【特許文献 4】

United States Patent 5,224,160

【特許文献 5】

United States Patent 6,006,328

【特許文献 6】

United States Patent 5,892,899

【特許文献 7】

United States Patent 6,178,509

50

【特許文献 8】

United States Patent 6,205,550

【0009】

【発明が解決しようとする課題】

しかしながら、上記従来技術は下記の問題を有している。すなわち、難読化されたプログラムは、時間をかければ解析される危険性がある。また、難読化の方式によってはその処理を計算機に行わせることが容易ではない。

【0010】

暗号化されたプログラムは人間が読んでも理解できないため解析が困難であるが、プログラムの実行前、若しくは、実行中に暗号化された命令コードは必ず復号されるため、復号後の命令コードが解析される危険性がある。

10

【0011】

また、暗号化された命令コードを復号するモジュールが解析、改ざんされると、暗号化によるプログラムの保護が容易に無効化されてしまう。また、ハードウェアで復号処理を行う場合は、解析される危険性は減るが、ソフトウェアのみを用いる場合と比べてコストが高む。

【0012】

断片化されたプログラムは、プログラム断片の実行を制御するモジュールが解析され、断片化によるプログラムの保護が容易に無効化されてしまう危険性がある。また、ハードウェアでプログラムの断片化の順序を制御すると解析される危険性は減るが、ソフトウェアのみを用いる場合と比べてコストが高む。また、難読化や暗号化と比べると自動化が困難であり、商品化が必ずしも容易ではない。

20

【0013】

本発明は、上記課題を解決するためになされたものであり、保護対象プログラムを解析及び改ざんが困難な自己書換えプログラムとして生成する自己書換え処理追加プログラム、自己書換え処理追加装置及び自己書換え処理追加方法を提供することを目的とする。

【0014】

【課題を解決するための手段】

本発明に係る自己書換え処理追加プログラムは、保護対象プログラムを構成する複数の命令コードのうちいずれか1カ所の命令コードを決定する命令コード決定手段、決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装する命令コード偽装手段、前記偽の命令コードを元の正しい命令コードに戻す第1のルーチンと当該第1のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度偽装する第2のルーチンとを生成するルーチン生成手段、前記第1及び第2のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成する自己書換えプログラム生成手段としてコンピュータを機能させ、前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第2のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第2のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第1のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第1のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第2のルーチンの追加位置として決定する。

30

40

【0015】

この構成によれば、保護対象プログラムは、プログラムを構成する命令コードうち1つの命令コードが対応する偽の命令コードで偽装される。そして、偽の命令コードを元の正しい命令コードに戻す第1のルーチンと、第1のルーチンによって戻された正しい命令コードを再度偽のコードで偽装する第2のルーチンとが生成される。そして、生成された第1

50

及び第2のルーチンがプログラム上の所定の位置に追加され自己書換え処理追加プログラムが生成される。

【0016】

この自己書換え処理プログラムを実行すると、第1のルーチンが実行されることにより、偽の命令コードは、偽装前の正しい命令コードに戻される。そして、正しい命令コードが実行された後、第2のルーチンが実行されることにより、その命令コードは再度、偽の命令コードで偽装される。すなわち、決定された命令コードは、実行前後の所定の期間のみ正しい命令コードが書き込まれ、それ以外の期間は偽の命令コードによって偽装された状態となる。そのため、保護対象プログラムの解析及び改ざんを困難にすることができる。また、この構成によれば、偽の命令コードは、実行される直前に確実に正しい命令コードに戻されるとともに、正しい命令コードは、実行された後、確実に偽の命令コードで偽装されるため、プログラムの誤動作を確実に防止することができる。そして、保護対象プログラムは、最終点に至るまでに確実に偽の命令コードによって偽装されることとなり、改ざん及び解読をさらに困難にすることができる。

10

【0017】

また、前記命令コード決定手段が前記命令コードを決定し、次いで前記命令コード偽装手段が前記命令コードに対応する偽の命令コードを決定し、次いで前記ルーチン生成手段が前記命令コードに対応する前記第1及び第2のルーチンを生成する処理を、複数回繰り返すことにより、複数の命令コードを決定し、前記複数の命令コードのそれぞれに対応する偽の命令コードを決定し、前記複数の命令コードのそれぞれに対応する前記第1及び第2のルーチンを生成することが好ましい。

20

【0018】

この構成によれば、複数の命令コードがそれぞれ対応する偽の命令コードに書換えられるため、保護対象となるプログラムの改ざん及び解析をより困難にすることができる。

【0023】

また、前記命令コード偽装手段は、前記正しい命令コードの機械語表現サイズと同一サイズの機械語表現を有する命令コードを前記偽の命令コードとして決定することが好ましい。

【0024】

この構成によれば、正しい命令コードと偽の命令コードとの機械語表現サイズが同一であるため、プログラム実行時に、偽の命令コードが確実に正しい命令コードに戻されることが可能となることに加えて、正しい命令コードが不自然な命令コードで偽装されることが防止され、保護対象のプログラムの解析及び改ざんをより困難にすることができる。

30

【0025】

また、前記偽の命令コードは、前記正しい命令コードの機械語表現のコード列のうち1バイトを置き換えた機械語表現を有する命令コードであることが好ましい。この構成によれば、偽の命令コードを正しい命令コードに戻す処理及び正しい命令コードを偽の命令コードに書換える処理を複数の命令を用いて表すことなく一つの命令で表現できるため、第1及び第2のルーチンのプログラムサイズを小さくすることができる。その結果、第1及び第2のルーチンが解析者によって発見されることが困難になるとともに、保護対象となるプログラムのサイズの増大を抑制することができる。

40

【0026】

また、前記第1のルーチンは、前記正しい命令コードの機械語表現のコード列と前記偽の命令コードの機械語表現のコード列との相違する箇所のみを前記偽の命令コードに対して書き込み、前記第2のルーチンは、前記正しい命令コードの機械語表現のコード列と前記偽の命令コードの機械語表現のコード列との相違する箇所のみを前記正しい命令コードに対して書き込むことが好ましい。この構成によれば、第1及び第2のルーチンの正しい命令コードと偽の命令コードとの相違する箇所のみを書換えるため、第1及び第2のルーチンの実行速度を早くすることができ、保護対象となるプログラムの実行速度の低下を抑制することができる。

50

【0027】

また、前記保護対象プログラムはアセンブリプログラムであり、前記自己書換えプログラムをアセンブルして機械語プログラムを得るアセンブル手段をさらに備えることが好ましい。この構成によれば、変更されたプログラムは、別途アセンブラを用いなくとも、本自己書換え処理追加プログラムにより、機械語プログラムを得ることができる。

【0028】

また、機械語プログラムを逆アセンブルする逆アセンブル手段をさらに備えることが好ましい。この構成によれば、保護対象プログラムが機械語プログラムであっても、逆アセンブルによって、機械語プログラムからアセンブリプログラムを得ることができ、そのアセンブリプログラムに対して、自己書換え処理を追加することにより、保護対象となるプログラムを解析及び改ざんが困難なプログラムにすることができる。

10

【0029】

また、前記アセンブリプログラムは、所定の高級言語で記述されたソースプログラムをコンパイルして得られたものであり、前記ソースプログラムをアセンブリプログラムにコンパイルして前記アセンブリプログラムを得るコンパイル手段をさらに備えることが好ましい。この構成によれば、保護対象となるプログラムが高級言語で記述されたプログラムであっても、解析及び改ざんが困難なプログラムにすることができる。

【0030】

また、機械語プログラムを連係編集するリンク手段をさらに備えることが好ましい。この構成によれば、ソースプログラムが複数のオブジェクトプログラムにコンパイルされた場合であっても、これらのオブジェクトプログラムの機械語プログラムが連係編集されるため、別途リンクを用いなくとも、実行可能プログラムを得ることができる。

20

【0032】

本発明に係る自己書換え処理追加装置は、保護対象プログラムを構成する複数の命令コードのうちいずれか1カ所の命令コードを決定する命令コード決定手段と、決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装する命令コード偽装手段と、前記偽の命令コードを元の正しい命令コードに戻す第1のルーチンと当該第1のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度偽装する第2のルーチンとを生成するルーチン生成手段と、前記第1及び第2のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成する自己書換えプログラム生成手段とを備え、前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第2のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第2のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第1のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第1のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第2のルーチンの追加位置として決定する。

30

40

【0033】

この構成によれば、保護対象プログラムは、プログラムを構成する命令コードの中から決定された1つの命令コードが対応する偽の命令コードで偽装される。そして、偽の命令コードを元の正しい命令コードに戻す第1のルーチンと、第1のルーチンによって戻された正しい命令コードを再度対応する偽のコードで偽装する第2のルーチンとが生成される。そして、生成された第1及び第2のルーチンがプログラム上の所定の位置に追加され自己書換え処理追加プログラムが生成される。

【0034】

この自己書換え処理プログラムを実行すると、第1のルーチンが実行されることにより、偽の命令コードは、元の正しい命令コードに戻される。そして、正しい命令コードが実行

50

された後、第2のルーチンが実行されることにより、その命令コードは再度、偽の命令コードで偽装される。すなわち、決定された命令コードは、実行前後の所定の期間のみ正しい命令コードが書き込まれ、それ以外の期間は偽の命令コードによって偽装された状態となる。そのため、保護対象プログラムの解析及び改ざんを困難にすることができる。

【0035】

本発明に係る自己書換え処理追加方法は、命令コード決定手段が、保護対象プログラムを構成する複数の命令コードのうちいずれか1カ所の命令コードを決定するステップ、命令コード偽装手段が、決定された命令コードに対応する、当該命令コードのサイズ以下のサイズである偽の命令コードを決定し、決定した偽の命令コードで前記命令コードを偽装するステップ、ルーチン生成手段が、前記偽の命令コードを元の正しい命令コードに戻す第1のルーチンと当該第1のルーチンによって戻された正しい命令コードを当該命令コードに対応する偽の命令コードで再度偽装する第2のルーチンとを生成するステップ、自己書換えプログラム生成手段が、前記第1及び第2のルーチンを前記保護対象プログラムの所定の位置に追加することにより自己書換えプログラムを生成するステップから構成され、前記自己書換えプログラム生成手段は、前記保護対象プログラムの開始点から前記偽の命令コードへ至るまでのフロー上の位置であって、かつ前記第2のルーチンから前記偽の命令コードへ至るまでのフロー上の位置であり、かつ前記第2のルーチンから終了点へ至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第1のルーチンの追加位置として決定すると共に、前記偽の命令コードから終了点へ至るまでのフロー上の位置であって、かつ、前記第1のルーチンから前記偽の命令コードに至るまでのフロー上の位置ではないという条件を満たす位置を示す複数の行の中からランダムに1つの行を前記第2のルーチンの追加位置として決定する。

【0036】

この構成によれば、保護対象プログラムは、プログラムを構成する命令コード中の1つの命令コードが対応する偽の命令コードで偽装される。そして、偽の命令コードを元の正しい命令コードに戻す第1のルーチンと、第1のルーチンによって戻された正しい命令コードを再度対応する偽のコードで偽装する第2のルーチンとが生成される。そして、生成された第1及び第2のルーチンがプログラム上の所定の位置に追加され自己書換え処理追加プログラムが生成される。

【0037】

この自己書換え処理プログラムを実行すると、第1のルーチンが実行されることにより、偽の命令コードは、正しい命令コードに戻される。そして、正しい命令コードが実行された後、第2のルーチンが実行されることにより、その命令コードは再度、偽の命令コードに書換えられる。すなわち、正しい命令コードは、実行前後の所定の期間のみプログラム上に書き込まれ、それ以外の期間は偽の命令コードによって偽装された状態となる。そのため、保護対象プログラムの解析及び改ざんを困難にすることができる。

【0038】

【発明の実施の形態】

図1は、本発明に係る自己書換え処理追加装置の構成を示すブロック図である。図1に示す自己書換え処理追加装置は、パーソナルコンピュータ等から構成され、入力装置1、ROM(リードオンリーメモリ)2、CPU(中央演算処理装置)3、RAM(ランダムアクセスメモリ)4、外部記憶装置5、表示装置6及び記録媒体駆動装置7を備える。各ブロックは内部のバスに接続され、このバスを介し各種データが各ブロック間で入出力され、CPU3の制御の下、各種処理が実行される。

【0039】

入力装置1は、キーボード、マウス等から構成され、オペレータが操作指示等を入力するために用いられる。ROM2には、システムプログラム等が予め記憶される。外部記憶装置5は、ハードディスクドライブ等から構成され、後述する自己書換え処理追加プログラム等を記憶している。

【0040】

10

20

30

40

50

CPU3は、外部記憶装置5から自己書換え処理追加プログラムを読み出し、読み出したプログラムを実行することで、命令コード決定手段、命令コード偽装手段、ルーチン生成手段、アセンブル手段、逆アセンブル手段、コンパイル手段及びリンク手段として機能する。

【0041】

RAM4は、CPU3の作業領域等として用いられる。表示装置6は、CRT（陰極線管）又は液晶表示装置等から構成され、CPU3の制御の下、各種画面を表示する。

【0042】

なお、自己書換え処理追加プログラムは、CD-ROM、フロッピィディスクドライブ等の記録媒体8に記録されており、この記録媒体8を媒体駆動装置7に装填してインストールすることにより、自己書換え処理追加プログラムを外部記憶装置5を格納させる。あるいは、自己書換え処理追加プログラムを通信ネットワークを介してダウンロードすることによりインストールして、外部記憶装置5に格納させてもよい。

10

【0043】

図2は、本自己書換え処理追加プログラムの処理を示すフローチャートである。

【0044】

処理を始めるにあたって、ユーザが作成等した高級言語のソースプログラム（保護対象プログラム）が、予めRAM4に記憶されているものとする。また、フローチャートに示すステップS3～S8のルーチンを繰り返す回数Nが予め設定されRAM4に記憶されているものとする。図3は、ソースプログラムの一例を示した図である。このソースプログラムは、ユーザにシリアル番号の入力を求め、それが正しければ（この例では、入力が123であれば）、先に続く処理に進むことができるが、正しくなければ、再びシリアル番号の入力を促すというプログラムである。

20

【0045】

ステップS1において、CPU3は、RAM4に記憶されたソースプログラムをコンパイルして、アセンブリプログラムを生成する。図4は、図3に示すソースプログラムをコンパイルすることにより得られたアセンブリプログラムの一部分を示した図である。

【0046】

ステップS2において、CPU3は、ステップS3～S9の処理を繰り返す回数をカウントするための変数nに0を代入し、変数nを初期化する。

30

【0047】

ステップS3において、CPU3は、偽の命令コードYによって偽装される命令コードである正しい命令コードyの決定を行う。本実施形態に用いられるアセンブリプログラムは、プログラムリスト上、1行につき1つの命令コードが記述されているものとする。したがって、CPU3は、プログラムリスト上の行をランダムに1つ選び、偽の命令コードYで偽装する位置P(Y)を決定する。これにより、正しい命令コードyが必然的に決定されることとなる。

【0048】

ステップS4において、CPU3は、正しい命令コードyをP(Y)に書き込む処理を行うルーチン（第1のルーチン）Xを挿入する位置P(X)及び偽の命令コードYをP(Y)に書き込む処理を行うルーチン（第2のルーチン）Zを挿入する位置P(Z)を決定する。

40

【0049】

詳細には、CPU3は、アセンブリプログラムに対して、次の条件1、2、3を満たすようにP(X)、P(Y)、P(Z)を決定する。

【0050】

（条件1）プログラム開始点からP(Y)へ至る制御フロー上にP(X)が存在する。

【0051】

（条件2）P(X)からP(Y)に至る制御フロー上にP(Z)が存在しない。

【0052】

50

(条件3) P(Z)からP(Y)に至る制御フロー上にP(X)が存在する。

【0053】

図5は、制御フロー上にP(X)、P(Y)、P(Z)が決定される様子を説明するための図である。CPU3は、まず、プログラムリスト上の行をランダムに1つ決定することによりP(Y)を決定する。次に、条件1を満たす位置を示す複数の行の中からランダムに1つの行を決定し、P(X)を決定する。次に、条件2、3を満たす位置を示す複数の行の中からランダムに1つの行を決定し、P(Z)を決定する。図5では、制御フロー上P(Y)よりも上流側の所定の位置がP(X)として決定されているとともに、制御フロー上P(Y)よりも下流側の所定の位置がP(Z)として決定されている。

【0054】

また、図4の例では、下から3行目がP(Y)として決定されているため、「`cmpl $123, -4(%ebp)`」が正しい命令コードyとなる。また、条件1を満たす位置であるプログラムの開始点からP(Y)までの位置を示す複数の行の中から1つの行(図4では上から4行目)が決定され、この次の行(図4では上から4行目と5行目の間の行)がP(X)の位置として決定される。また、条件2及び3を満たす位置の中から1つの行が決定され(図4では下から3行目)、この次の行(図4では下から3行目と2行目の間の行)、がP(Z)として決定される。

【0055】

ステップS5において、CPU3は、正しい命令コードyを偽装するための偽の命令コードYを決定する。この場合、CPU3は、正しい命令コードyの機械語表現のバイト数と同一バイト数を有し、かつ、1バイトだけ異なる機械語表現を有する命令コードの中からランダムに1つの命令コードを決定することにより偽の命令コードYを決定する。

【0056】

ステップS6において、CPU3は、ルーチンX及びZを生成する。そして、ステップS7において、ルーチンXをP(X)に挿入するとともに、ルーチンZをP(Z)に挿入する。加えて、正しい命令コードyを偽の命令コードYに変更する。これにより、正しい命令コードyは、偽の命令コードYで偽装される。図6は、図4に示すアセンブリプログラムに対してルーチンX及びZを挿入するとともに、正しい命令コードyを偽の命令コードYに変更されたアセンブリプログラムを示している。図6に示すように、P(Y)には、「`cmpl $123, -4(%ebp)`」が偽の命令コードである「`or $123, -4(%ebp)`」に変更されている。 「`cmpl $123, -4(%ebp)`」の16進数による機械語表現は「83 7D FC 7B」であり、4バイトのサイズを有する。一方、「`or $123, -4(%ebp)`」の16進数による機械語表現は、「83 4D FC 7B」である。したがって、偽の命令コードYとして決定された「`or $123, -4(%ebp)`」は、元の命令コード「`cmpl $123, -4(%ebp)`」に対して、その機械語表現が同一バイト数を有し、かつ、1バイトのみ異なるという条件を満たしている。

【0057】

プログラムの実行時には、各命令コード(の機械語表現)はコンピュータのメモリ上に存在しているため、書換え前の正しい命令コードyと書換え後の偽の命令コードYのサイズが異なると問題が生じる。すなわち、偽の命令コードYが正しい命令コードyよりも大きい場合、正しい命令コードyを偽の命令コードYに書換えようとしても、正しい命令コードyの位置には当該命令コードのサイズ分しかメモリが空いておらず、偽の命令コードYに書換えることができない。したがって、この場合、正しい命令コードyを偽の命令コードYに書換えることが原理的に不可能となる。一方、正しい命令コードyが偽の命令コードYよりも大きい場合は、書換えは不可能ではないが、余った部分のメモリに無意味な命令を入れるなどする必要があり、偽の命令コードYが不自然な命令コードとなってしまうおそれがあり好ましくない。

【0058】

したがって、偽の命令コードYとしては、正しい命令コードyに対して、その機械語表現が同一サイズであるものが好ましいのである。しかしながら、「正しい命令コードyのサイズが偽の命令コードYのサイズよりも大きい」場合は、原理的に書換えることが可能で

10

20

30

40

50

あるため、正しい命令コード y よりもサイズの小さい命令コードを偽の命令コード Y としてもよい。この場合、上述したように余った部分に無意味な命令を入れる必要がある。

【 0 0 5 9 】

また、多くのコンピュータでは、メモリ上のデータを書換える際の最小単位が 1 バイトである。そのため、1 バイトだけの書換えで正しい命令コード y から偽の命令コード Y に（又は偽の命令コード Y から正しい命令コード y に）変換できると、偽の命令コード Y を正しい命令コード y へ（又は正しい命令コード y を偽の命令コード Y へ）変換するための処理が一つの命令で表現できることとなる。そうすると、その処理を含むルーチン X 及び Z のサイズが小さくなるため、解析者は、ルーチン X 及び Z を発見することが困難となる。加えて、自己書換えプログラム全体のサイズの増大が抑制される。したがって、偽の命令コード Y としては、正しい命令コード y に対して、その機械語表現が 1 バイトのみ異なる命令コードが好ましいのである。なお、正しい命令コード y と偽の命令コード Y とは、その機械語表現が 1 バイトのみ異なるものに限定されず、1 ビット、や 4 ビットのみ異なるというようなものであってもよい。

10

【 0 0 6 0 】

次に、図 6 に示すルーチン X について説明する。

【 0 0 6 1 】

`pushl %eax` : 汎用レジスタ `eax` の値をスタックに退避する。

【 0 0 6 2 】

`movl $CFLOC_1, %eax` : レジスタ `eax` にラベル `CFLOC_1` の値を代入する。これにより、偽の命令コード Y のプログラム上の位置がレジスタ `eax` に格納される。

20

【 0 0 6 3 】

`movb $125, 1(%eax)` : レジスタ `eax` に 1 バイト加えた値が指すプログラム上の位置、すなわち偽の命令コード Y の 2 バイト目に 1 2 5 (1 6 進数では 7 D) を書き込む。これにより、偽の命令コード Y が正しい命令コード y に書き換わる。ここでの偽の命令コード Y は「`or $123, -4(%ebp)`」であり、「83 4D FC 7B」という機械語で表される。レジスタ `eax` の値はこの命令の先頭 (8 3) を指しており、`eax` に 1 を加えた値はこの命令の 2 バイト目 (4 D) を指すことになる。この 4 D を 7 D に書換えると、「83 7D FC 7B」という機械語で表される `cmpl $123, -4(%ebp)` という命令になる。

`popl %eax` : 汎用レジスタ `eax` の値をスタックから復元する。

30

なお、ルーチン X では、ルーチンの開始時に「`pushl %eax`」を実行し、終了時に「`popl %eax`」を実行しているため、たとえルーチン内で `eax` の値が変更されたとしても、ルーチン終了時の段階では、`eax` には、ルーチン開始時のものと同じ値が格納されることとなりプログラムの誤動作が防止される。

【 0 0 6 4 】

次に、図 6 に示すルーチン Z について説明する。

【 0 0 6 5 】

`pushl %eax` : 汎用レジスタ `eax` の退避を行う。

【 0 0 6 6 】

`movl $CFLOC_1, %eax` : レジスタ `eax` にラベル `CFLOC_1` の値を代入する。これにより、偽の命令コード Y のプログラム上の位置がレジスタ `eax` に格納される。

40

【 0 0 6 7 】

`movb $77, 1(%eax)` : レジスタ `eax` に 1 バイト加えた値が指すプログラム上の位置、すなわち正しい命令コード y (この時点ではルーチン X によって偽の命令コード Y が正しい命令コード y に書換えられている) の 2 バイト目に 7 7 (1 6 進数では 4 D) を書き込む。これにより正しい命令コード y が再び偽の命令コード Y となる。

【 0 0 6 8 】

`popl %eax` : 汎用レジスタ `eax` の復元を行う。

【 0 0 6 9 】

なお、ルーチン Z は、ルーチン X 同様、ルーチンの開始時に「`pushl %eax`」を実行し、終

50

了時に「popl %eax」を実行しているため、たとえルーチン内でeaxの値が変更されたとしても、ルーチン終了時の段階では、eaxには、ルーチン開始時のものと同じ値が格納されていることとなりプログラムの誤動作が防止される。

【 0 0 7 0 】

ステップ S 8 において、CPU 3 は、ルーチン X 及び Z が間接的に P (Y) を指定するように、ルーチン X 及び Z に対して複雑化処理を施す。ルーチン X 及び Z はどちらも P (Y) に存在する命令コードを書換えるルーチンであるため、P (Y) を指し示す命令コードが含まれている。そのため、解析者は、プログラム中から、同じアドレスを指し示す 2 つの命令コードを探し出すことにより、ルーチン X とルーチン Z とを発見し、P (Y) を見つける危険性がある。そこで、CPU 3 は、ルーチン X 及び Z に直接 P (Y) を指し示す命令コードが含まれないように、ルーチン X 及び Z が間接的に P (Y) を指定するようにルーチン X 及び Z に複雑化処理を施し、保護対象プログラムの解析及び改ざんをより困難にしている。

10

【 0 0 7 1 】

図 7 は、図 6 のアセンブリプログラムに対してルーチン X 及び Z に複雑化処理が施したアセンブリプログラムを示している。図 7 及び図 6 を比較すれば分かるように、図 6 では、ルーチン X 及び Z 内には、ともに P (Y) を指し示すラベル「CFLOC_1」が記載されているが、図 7 では、P (Y) を指し示すラベル「CFLOC_1」が記載されていない。そのため、解析者は、P (Y) を見つけることが困難となっている。

【 0 0 7 2 】

次に、図 7 に示すルーチン X について説明する。

20

【 0 0 7 3 】

pushf : フラグレジスタの退避を行う。後述のaddl やsubl といった命令はフラグレジスタと呼ばれるレジスタを変化させるため、汎用レジスタeaxを退避させることと同じ理由で、フラグレジスタも退避させておくのである。

【 0 0 7 4 】

pushl %eax : 汎用レジスタeaxの退避を行う。

【 0 0 7 5 】

movl \$A2, %eax : レジスタeaxにラベル A 2 の値を代入する。

【 0 0 7 6 】

subl \$20, %eax : レジスタeaxに格納されている値を 1 0 進数で 2 0 減算した値にする。この時点で、eaxに格納されている値が偽の命令コード Y のプログラム上の位置を指すものとなる。

30

【 0 0 7 7 】

movb \$125, 1(%eax) : 図 6 の X の説明で述べたものと同じである。偽の命令コード Y を正しい命令コード y に書換える処理である。

【 0 0 7 8 】

popl %eax : 汎用レジスタeaxの復元を行う。

【 0 0 7 9 】

popf : フラグレジスタの復元を行う。

40

【 0 0 8 0 】

次に、図 7 に示すルーチン Z について説明する。

【 0 0 8 1 】

pushf : フラグレジスタの退避を行う。

【 0 0 8 2 】

pushl %eax : 汎用レジスタeaxの退避を行う。

【 0 0 8 3 】

movl \$A1, %eax : レジスタeaxにラベル A 1 の値を代入する。

【 0 0 8 4 】

addl \$23, %eax : レジスタeaxに格納されている値を 1 0 進数で 2 3 加算した値にする。

50

この時点で、eaxに格納されている値が正しい命令 y (この時点ではルーチン X によって偽の命令コード Y が正しい命令コード y に書換えられている) のプログラム上の位置を指すものとなる。

【 0 0 8 5 】

movb \$77, 1(%eax) : 図 6 のルーチン Z の説明で述べたものと同じである。正しい命令コード y を偽の命令コード Y に書換える処理である。

【 0 0 8 6 】

popl %eax : 汎用レジスタ eax の復元を行う。

【 0 0 8 7 】

popf : フラグレジスタの復元を行う。

10

【 0 0 8 8 】

ステップ S 9 において、CPU 3 は、変数 n に対して 1 を加算する。ステップ S 10 において、CPU 3 は、変数 n が $n \geq N$ の条件を満たすか否かの判断を行い、変数 n がこの条件を満たさない場合 (ステップ S 10 で NO) ステップ S 3 に戻る。一方、変数 n が $n \geq N$ の条件を満たす場合 (ステップ S 10 で YES)、ステップ S 11 に進む。これによりステップ S 3 ~ S 8 の処理が N 回繰り返して実行され、N 個の正しい命令コード y がそれぞれに対して決定された偽の命令コード Y で偽造化された自己書換えプログラムが生成される。なお、N の値を大きく設定すると、偽造化される命令コードの数が増大するため、生成された自己書換えプログラムの解読がより困難となる。図 8 は、アセンブリプログラムにステップ S 3 ~ S 8 の処理を N 回繰り返すことにより生成された自己書換えプログラムを示している。図 8 に示すように、自己書換えプログラムには、元のアセンブリプログラムに対して、複数 (N 個) のルーチン X 及び Z が挿入されているとともに、複数 (N 個) の偽の命令コード Y が書き込まれている。

20

【 0 0 8 9 】

ステップ S 11 において、CPU 3 は、自己書換えプログラムをアSEMBルすることにより自己書換えプログラムの機械語プログラムを生成する。この場合必要であれば、CPU 3 は、生成された機械語プログラムをリンク (連係編集) して実行可能プログラムを生成する。さらに、CPU 3 は、生成した実行可能プログラムに対して、そのプログラムのコード領域への書き込みを許可するフラグを立てる等の処理を施し、実行時にその実行可能プログラム自身が、自らのコード領域を書き込むことができるようにする。

30

【 0 0 9 0 】

次に自己書換えプログラムが CPU 3 によって実行される様子を説明する。まず、開始点から順に種々の命令コードが実行されていき、P (X) に到達すると、ルーチン X が実行され、P (Y) に記載された偽の命令コード Y が正しい命令コード y に書換えられる。そして、P (X) 以降の種々の命令コードが実行されていき、P (Y) に到達すると、正しい命令コード y が実行される。そして、P (Y) 以降の種々の命令コードが実行され、P (Z) に到達すると、ルーチン Z が実行され、P (Y) に書き込まれている正しい命令コード y が再度偽の命令コード Y に書換えられる。

【 0 0 9 1 】

そのため、P (Y) には、実行前後の所定の期間のみ正しい命令コード y が書き込まれ、それ以外の期間は、偽の命令コード Y によって偽装された状態となる。その結果、プログラムの解析及び改ざんを困難にすることができる。

40

【 0 0 9 2 】

また、本自己書換え処理追加プログラムによれば、複数の命令コードのそれぞれが偽の命令コード Y で偽装化されるため、保護対象プログラムの解析及び改ざんをより困難にすることができる。

【 0 0 9 3 】

なお、上記実施形態では、自己書換え処理追加プログラムは、アセンブリプログラムを元に自己書換えプログラムを生成したが、これに限定されず、C 言語、FORTRAN、PASCAL 等の高級言語で書かれたプログラムを元に自己書換えプログラムを生成しても

50

よい。

【 0 0 9 4 】

また、上記実施形態では、自己書換え処理追加プログラムは、高級言語のソースプログラムをコンパイルすることにより、アセンブリプログラムを取得し、このアセンブリプログラムを元に自己書換えプログラムを生成しているが、これに限定されず、アセンブリで作成されたアセンブリプログラム、あるいは他のコンパイラによって得られたアセンブリプログラムを元に、自己書換えプログラムを生成してもよい。この場合、自己書換え処理追加プログラムは、コンパイル手段が不要となる。

【 0 0 9 5 】

また、上記実施形態では、自己書換え処理追加プログラムは、生成した自己書換えプログラムをアセンブルして機械語プログラムを得ていたが、これに限定されず、生成した自己書換えプログラムを他のアセンブラを用いてアセンブルしてもよい。この場合、自己書換え処理追加プログラムは、アセンブル手段が不要となる。

【 0 0 9 6 】

また、上記実施形態では、プログラムの行をランダムに選ぶことにより P (Y)、P (X)、P (Z) を決定したがこれに限定されず、他の手法を用いても決定してもよい。

【 0 0 9 7 】

【 発明の効果 】

本発明によれば、保護対象のプログラムを構成する正しい命令コードが偽の命令コードによって偽装化されるとともに、実行前後の所定の期間のみ、偽の命令コードが正しい命令コードに書換えられ、それ以外の期間は、偽の命令コードによって偽造化された状態となるため、プログラムの解析及び改ざんを困難にすることができる。

【 図面の簡単な説明 】

【 図 1 】 本発明に係る自己書換え処理追加装置の構成を示すブロック図である。

【 図 2 】 本自己書換え処理追加プログラムの処理を示すフローチャートである。

【 図 3 】 ソースプログラムの一例を示した図である。

【 図 4 】 図 3 に示すソースプログラムをコンパイルすることにより得られたアセンブリプログラムを示した図である。

【 図 5 】 制御フロー上に P (X)、P (Y)、P (Z) の位置が決定される様子を説明するための図である。

【 図 6 】 図 4 に示すアセンブリプログラムに対してルーチン X 及び Z を挿入するとともに、正しい命令コード y を偽の命令コード Y に書換えたアセンブリプログラムを示している。

【 図 7 】 図 6 のアセンブリプログラムに対してルーチン X、Z が間接的に P (Y) を指定するように複雑化処理が施されたアセンブリプログラムを示している。

【 図 8 】 保護対象となるプログラムにステップ S 3 ~ S 8 の処理を繰り返し実行することにより、生成された自己書換えプログラムを示している。

【 図 9 】 典型的なプログラムの難読化の仕組みを示した図である。

【 図 1 0 】 典型的なプログラムの暗号化の仕組みを示した図である。

【 図 1 1 】 典型的なプログラムの断片化の仕組みを示した図である。

【 符号の説明 】

- 1 入力装置
- 2 ROM
- 3 CPU
- 4 RAM
- 5 外部記憶装置
- 6 表示装置
- 7 記録媒体駆動装置
- 8 記録媒体
- y 正しい命令コード

10

20

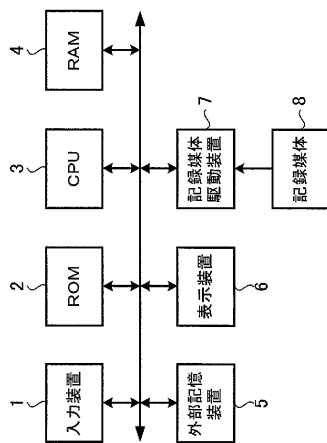
30

40

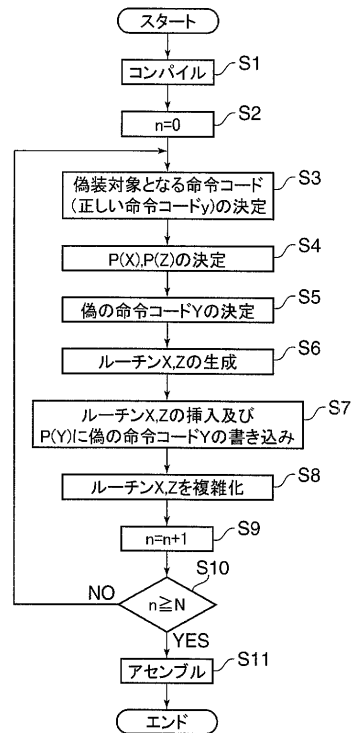
50

- Y 偽の命令コード
- X ルーチン (第1のルーチン)
- Z ルーチン (第2のルーチン)
- P (X) ルーチン X が挿入される位置
- P (Y) 偽の命令コードが書き込まれる位置
- P (Z) ルーチン Z が挿入される位置

【 図 1 】



【 図 2 】



【 図 3 】

```

int main(void) {
    long n;

    while(1) {
        printf ("Enter Your Serial Number: ");
        scanf ("%ld",&n);

        if (n = 123) {
            break;
        } else {
            printf ("Wrong! ≠ n");
        }
    }
    printf ("Correct ≠ n");
    : (中略)
    return 0;
}

```

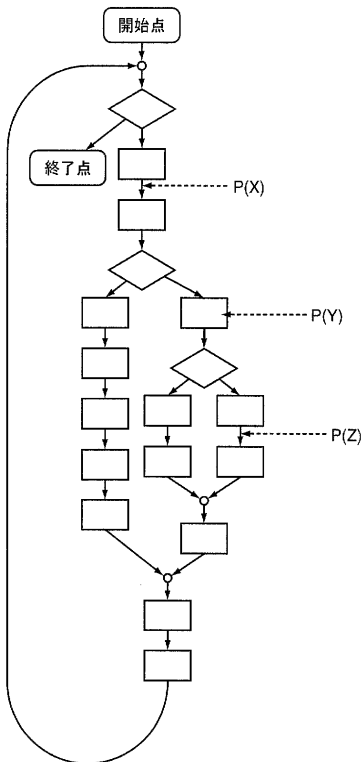
【 図 4 】

```

:
pushl %ebp
movl %esp,%ebp
subl $24,%esp
call __main
L5:
addl $-12,%esp
pushl $LC0
call _printf
addl $16,%esp
addl $-8,%esp
leal -4(%ebp),%eax
pushl %eax
pushl $LC1
call _scanf
addl $16,%esp
cmpl $123,-4(%ebp)
jne L6
jmp L4
:

```

【 図 5 】



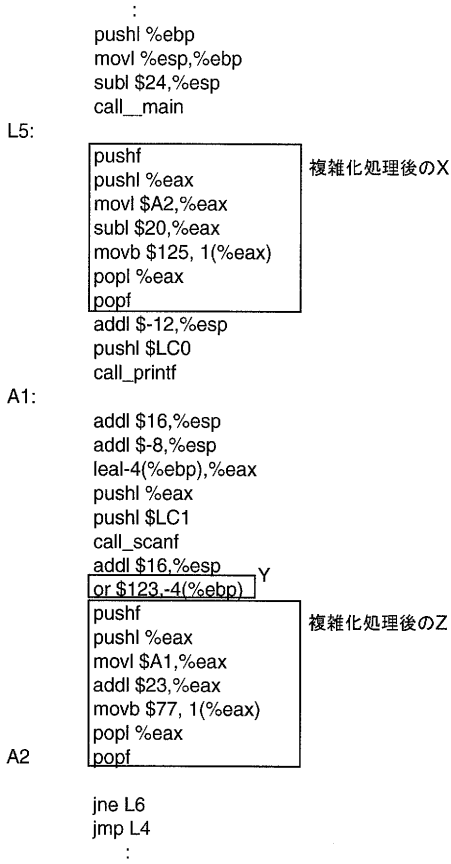
【 図 6 】

```

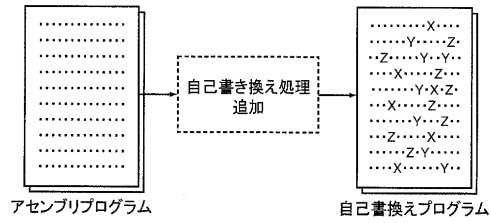
:
pushl %ebp
movl %esp,%ebp
subl $24,%esp
call __main
L5:
pushl %eax
movl $CFLOC_1, %eax
movb $125, 1(%eax)
popl %eax
addl $-12,%esp
pushl $LC0
call _printf
addl $16,%esp
addl $-8,%esp
leal -4(%ebp),%eax
pushl %eax
pushl $LC1
call _scanf
addl $16,%esp
CFLOC_1:
or $123,-4(%ebp)
pushl %eax
movl $CFLOC_1, %eax
movb $77, 1(%eax)
popl %eax
jne L6
jmp L4
:

```

【 図 7 】



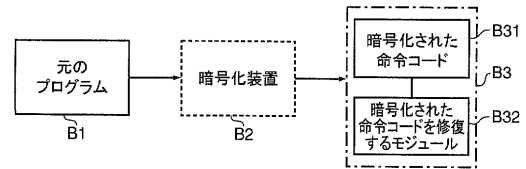
【 図 8 】



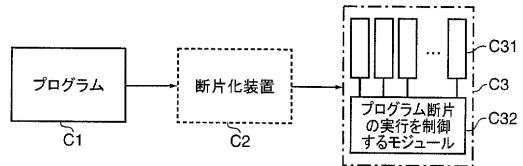
【 図 9 】



【 図 10 】



【 図 11 】



フロントページの続き

(72)発明者 神崎 雄一郎
奈良県生駒市高山町8916-5 奈良先端科学技術大学院大学内

審査官 本郷 彰

(56)参考文献 特開平11-272563(JP,A)
特開昭61-190630(JP,A)
特開2002-244989(JP,A)
特開2002-132364(JP,A)
特開2002-288042(JP,A)

(58)調査した分野(Int.Cl., DB名)
G06F 21/22
G06F 21/24