(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2004-192068
(P2004-192068A)
(43)公開日　平成16年7月8日(2004.7.8)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 1/00	G06F	9/06 660L	5B076

審査請求 未請求　請求項の数 14　OL　（全 18 頁）

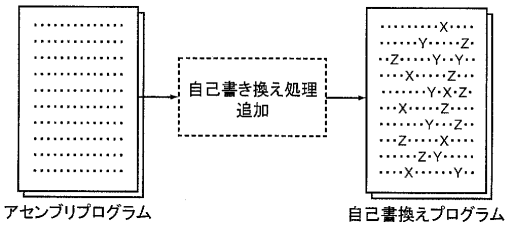
(21)出願番号　特願2002-355881(P2002-355881)
(22)出願日　平成14年12月6日(2002.12.6)

(71)出願人　500466739
財団法人奈良先端科学技術大学院大学支援財団
奈良県生駒市高山町８９１６番地１２
(74)代理人　100067828
弁理士　小谷　悦司
(74)代理人　100075409
弁理士　植木　久一
(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
(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

Fターム(参考)　5B076 FA01

(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)【要約】
【課題】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する。
【解決手段】アセンブリプログラムを構成する複数の命令コードを偽の命令コードＹで偽装するとともに、偽の命令コードＹを元の正しい命令コードｙに戻すルーチンＸと、ルーチンＸが戻した正しい命令コードｙを再度偽の命令コードＹで偽装するルーチンＺとを生成し、生成したルーチンＸ及びＺをアセンブリプログラム上の所定の位置に書き込み、自己書換えプログラムを生成する。
【選択図】　図８



【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命令コードの中から１つ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は、複数の命令コードを決定し、
前記命令コード偽装手段は、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、
前記ルーチン生成手段は、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】
前記自己書換えプログラム生成手段は、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所定の位置に前記第１のルーチンが存在し、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所定の位置に前記第２のルーチンが存在せず、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所定の位置に前記第１のルーチンが存在するという条件を満たすように前記第１及び第２のルーチンの追加位置を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】
前記自己書換えプログラム生成手段は、請求項３記載の条件に加えて、さら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所定の位置に前記第２のルーチンが存在するという条件を満たすように前記第１及び第２のルーチンの追加位置を決定す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１～４のいずれかに記載の自己書換え処理追加プログラム。
【請求項６】
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５記載の自己書換え処理追加プログラム。
【請求項７】
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１～６記載のいずれかに記載の自己書換え処理追加プログラム。
【請求項８】
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１～７のいずれかに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８記載の自己書換え処理追加プログラム。
【請求項１０】
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８又は９記載の自己書換え処理追加プログラム。
【請求項１１】
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１０記載の自己書換え処理追加プログラム。
【請求項１２】
前記ルーチン生成手段は、前記偽の命令コードで偽装されたプログラム上の位置を間接的に指定するように前記第１及び第２のルーチンに複雑化処理を施すことを特徴とする請求項１～１１のいずれかに記載の自己書換え処理追加プログラム。
【請求項１３】
保護対象プログラムを構成する命令コードの中から１つ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ることを特徴とする自己書換え処理追加装置。
【請求項１４】
保護対象プログラムを構成する命令コードの中から１つの命令コードを決定するステップと、
決定された命令コードに対応す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と、
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ステップとを備えることを特徴とする自己書換え処理追加方法。
【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。

【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Ａ１は、難読化装置Ａ２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Ａ３として出力される。難読化されたプログラムＡ３は、元のプログラムＡ１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Ａ３が配布された使用者（ユーザ）や第三者は、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Ａ３は、元のプログラムＡ１と同様、計算機上でそのまま実行することができる。

【０００４】
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。

【０００５】
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Ｂ１は、暗号化装置Ｂ２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Ｂ３として出力される。暗号化されたプログラムＢ３は、暗号化された命令コードＢ３１及び暗号化された命令コードＢ３１を復号するためのモジュールＢ３２から構成されている。

【０００６】
暗号化された命令コードＢ３１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Ｂ３１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Ｂ３１を復号するためのモジュールＢ３２が必要となる。このモジュールＢ３２は、暗号化されたプログラムＢ３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Ｂ３２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。

【０００７】
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Ｃ１は、断片化装置Ｃ２によって、所定の処理が施され、断片化されたプログラムＣ３として出力される。断片化されたプログラムＣ３は、分割された多数のプログラム断片Ｃ３１及びそれらの実行を制御するモジュールＣ３２から構成されている。人間は、個々プログラム断片Ｃ３１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。

【０００８】
【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．
【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－Ｉ、Ｎｏ７、ｐｐ６４４－６５

2、July1997.
【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 inComputer 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.
【非特許文献5】
M.mambo,T.Murayama,and E.Okamoto,”A tentative approach to constructingtampei－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（POPL98）,San Diego,California,1998.
【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
【非特許文献１１】
D.J.Albert and S.P.Morse,”Combating software piracy by encryption and key manegement”,Computer,April 1982

【非特許文献１２】
石間宏之、斎藤和夫、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
【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
【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
【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。
【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が、ソフトウェアのみを用いる場合と比べてコストが嵩む。
【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嵩む。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。
【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命令コードの中から１つ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ることを特徴とする。

【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。

【００１７】

また、前記命令コード決定手段は、複数の命令コードを決定し、前記命令コード偽装手段は、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、前記ルーチン生成手段は、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。

【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。

【００１９】

また、前記自己書換えプログラム生成手段は、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所定の位置に前記第１のルーチンが存在し、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所定の位置に前記第２のルーチンが存在せず、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所定の位置に前記第１のルーチンが存在するという条件を満たすように前記第１及び第２のルーチンの追加位置を決定することが好ましい。

【００２０】

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。

【００２１】

また、前記自己書換えプログラム生成手段は、請求項３記載の条件に加えてさら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所定の位置に前記第２のルーチンが存在するという条件を満たすように、前記第１及び第２のルーチンの追加位置を決定すること

が好ましい。
【００２２】
この構成によれば、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。
【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。
【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。
【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。
【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。
【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、別途リンカを用いなくとも、実行可能プログラムを得ることができる。

【００３１】

また、前記ルーチン生成手段は、前記偽の命令コードで偽装したプログラム上の位置を間接的に指定するように前記第１及び第２のルーチンに書換え処理を施すことが好ましい。この構成によれば、第１及び第２のルーチン内に、偽の命令コードで偽装したプログラム上の位置が直接現れないこととなり、プログラムの改ざん及び解析がさらに困難にすることができる。

【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命令コードの中から１つ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ることを特徴とする。

【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、保護対象プログラムを構成する命令コードの中から１つの命令コードを決定するステップと、決定された命令コードに対応す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決定された命令コードを偽装するステップと、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ステップとを備えることを特徴とする。

【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】

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。

【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。

【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。

【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。

【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】

（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。

【００５９】

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。

【００６０】

次に、図６に示すルーチンＸについて説明する。

【００６１】

ｐｕｓｈｌ　％ｅａｘ：汎用レジスタｅａｘの値をスタックに退避する。

【００６２】

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。

【００６３】

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１２５（１６進数では７Ｄ）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（８３）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（４Ｄ）を指すことになる。この４Ｄを７Ｄに書換えると、「８３　７Ｄ　ＦＣ　７Ｂ」という機械語で表されるｃｍ

ｐｌ　＄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。
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。
【００６４】
次に、図６に示すルーチンＺについて説明する。
【００６５】
ｐｕｓｈｌ　％ｅａｘ：汎用レジスタｅａｘの退避を行う。
【００６６】
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。
【００６７】
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７７（１６進数では４Ｄ）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。
【００６８】
ｐｏｐｌ　％ｅａｘ：汎用レジスタｅａｘの復元を行う。
【００６９】
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。
【００７０】
ステップＳ８において、ＣＰＵ３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、ＣＰＵ３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。
【００７１】
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。
【００７２】
次に、図７に示すルーチンＸについて説明する。
【００７３】
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。
【００７４】
ｐｕｓｈｌ　％ｅａｘ：汎用レジスタｅａｘの退避を行う。

【００７５】
ｍｏｖｌ　＄Ａ２，　％ｅａｘ：レジスタｅａｘにラベルＡ２の値を代入する。
【００７６】
ｓｕｂｌ　＄２０，　％ｅａｘ：レジスタｅａｘに格納されている値を１０進数で２０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。
【００７７】
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。
【００７８】
ｐｏｐｌ　％ｅａｘ：汎用レジスタｅａｘの復元を行う。
【００７９】
ｐｏｐｆ：フラグレジスタの復元を行う。
【００８０】
次に、図７に示すルーチンＺについて説明する。
【００８１】
ｐｕｓｈｆ：フラグレジスタの退避を行う。
【００８２】
ｐｕｓｈｌ　％ｅａｘ：汎用レジスタｅａｘの退避を行う。
【００８３】
ｍｏｖｌ　＄Ａ１，　％ｅａｘ：レジスタｅａｘにラベルＡ１の値を代入する。
【００８４】
ａｄｄｌ　＄２３，　％ｅａｘ：レジスタｅａｘに格納されている値を１０進数で２３加算した値にする。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。
【００８５】
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。
【００８６】
ｐｏｐｌ　％ｅａｘ：汎用レジスタｅａｘの復元を行う。
【００８７】
ｐｏｐｆ：フラグレジスタの復元を行う。
【００８８】
ステップＳ９において、ＣＰＵ３は、変数ｎに対して１を加算する。ステップＳ１０において、ＣＰＵ３は、変数ｎがｎ＞＝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Ｓ１０でＮＯ）ステップＳ３に戻る。一方、変数ｎがｎ＞＝Ｎの条件を満たす場合（ステップＳ１０でＹＥＳ）、ステップＳ１１に進む。これによりステップＳ３～Ｓ８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Ｓ３～Ｓ８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。
【００８９】
ステップＳ１１において、ＣＰＵ３は、自己書換えプログラムをアセンブ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、ＣＰＵ３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、ＣＰＵ３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。

【００９０】

次に自己書換えプログラムがＣＰＵ３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。

【００９１】

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。

【００９２】

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。

【００９３】

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よい。

【００９４】

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。

【００９５】

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。

【００９６】

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。

【００９７】

【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。

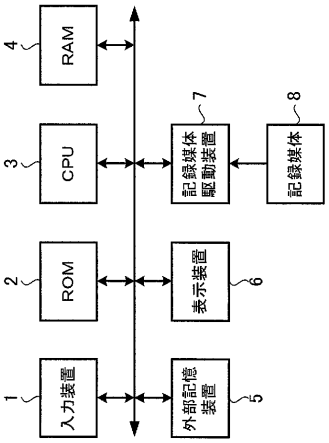
【図２】本自己書換え処理追加プログラムの処理を示すフローチャートである。

【図３】ソースプログラムの一例を示した図である。

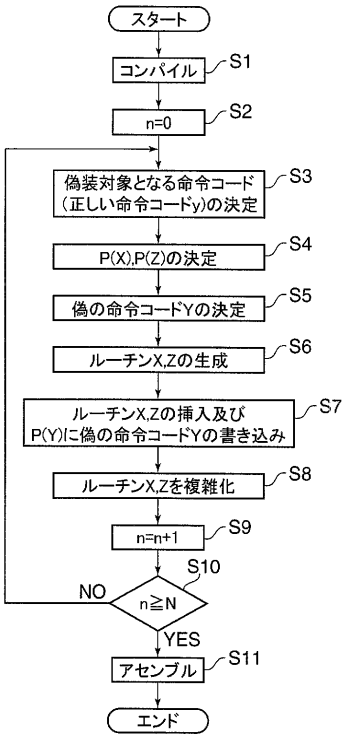
【図４】図３に示すソースプログラムをコンパイルすることにより得られたアセンブリプ

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Ｓ３～Ｓ８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１０】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１１】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
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】



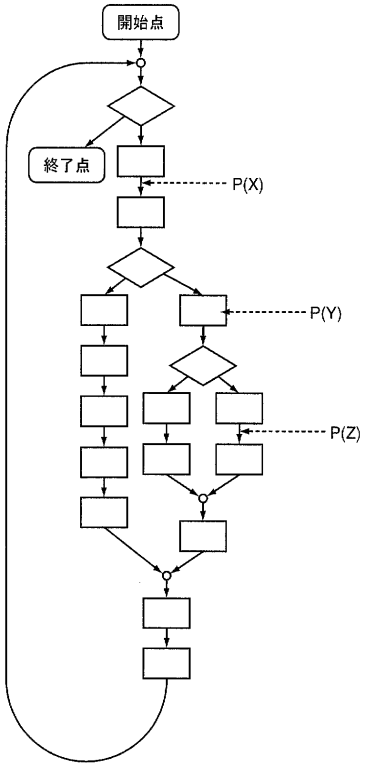
【図３】

```
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                    P(X)
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)            P(Y)
    jne L6
    jmp L4                        P(Z)
        :
```

【図５】



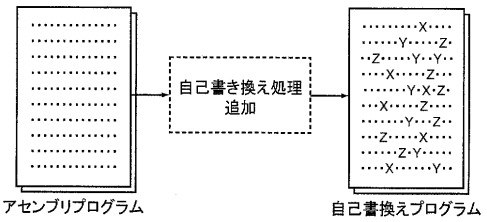
【図６】

```
        :
        pushl %ebp
        movl %esp,%ebp
        subl $24,%esp
        call__main
L5:
        pushl %eax
        movl $CFLOC_1, %eax      X
        movb $125, 1(%eax)       ←P(X)
        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:                         Y
        or $123,-4(%ebp)         ←P(Y)
        pushl %eax
        movl $CFLOC_1, %eax      Z
        movb $77, 1(%eax)        ←P(Z)
        popl %eax
        jne L6
        jmp L4
        :
```

【図７】

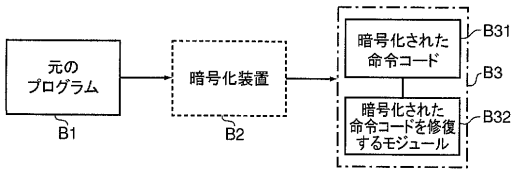
```
        :
        pushl %ebp
        movl %esp,%ebp
        subl $24,%esp
        call__main
L5:
        pushf                    複雑化処理後のX
        pushl %eax
        movl $A2,%eax
        subl $20,%eax
        movb $125, 1(%eax)
        popl %eax
        popf
        addl $-12,%esp
        pushl $LC0
        call_printf
A1:
        addl $16,%esp
        addl $-8,%esp
        leal-4(%ebp),%eax
        pushl %eax
        pushl $LC1
        call_scanf
        addl $16,%esp
        or $123,-4(%ebp)         Y
        pushf                    複雑化処理後のZ
        pushl %eax
        movl $A1,%eax
        addl $23,%eax
        movb $77, 1(%eax)
        popl %eax
A2      popf
        jne L6
        jmp L4
        :
```

【図８】



【図９】

【図１０】



【図１１】