(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4892167号
(P4892167)
(45)発行日　平成24年3月7日(2012.3.7)　(24)登録日　平成23年12月22日(2011.12.22)

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

請求項の数 35 (全 24 頁)

(21)出願番号	特願2002-512772 (P2002-512772)
(86)(22)出願日	平成13年7月13日 (2001.7.13)
(65)公表番号	特表2004-511031 (P2004-511031A)
(43)公表日	平成16年4月8日 (2004.4.8)
(86)国際出願番号	PCT/GB2001/003159
(87)国際公開番号	WO2002/006925
(87)国際公開日	平成14年1月24日 (2002.1.24)
審査請求日	平成20年7月10日 (2008.7.10)
(31)優先権主張番号	0017481.3
(32)優先日	平成12年7月18日 (2000.7.18)
(33)優先権主張国	英国 (GB)
(31)優先権主張番号	0102982.6
(32)優先日	平成13年2月7日 (2001.2.7)
(33)優先権主張国	英国 (GB)

前置審査

(73)特許権者　506081792
シンプレックス　メジャー　センドリアン　ベルハッド
マレーシア国　スランゴール　ディーイー　ペタリン　ジャヤ　４７４００　ジャラン　エスエス　２０－２７　ダマンサラ　イタン　１　セカンド　フロアー　ブロック　シー　ユニット　２０３
(74)代理人　100083149
弁理士　日比　紀彦
(72)発明者　サファ　ジョン　アラム
イギリス国　ノッティンガム　エヌジー７　１ディーユー　ザ　パーク　エステイト　レントン　ロード　３４

最終頁に続く

(54)【発明の名称】ディジタル・データ保護構成

(57)【特許請求の範囲】

【請求項１】

実行可能コードを包含するディジタル・データ保護装置であって、前記実行可能コードは、実行されたときに、被保護データを使用可能な形式で含む別のコードを生成することが可能になる、前記被保護データに関連する充分な情報を組み込んでいるとともに、さらに前記被保護データの改ざんを検出するべく動作可能なセキュリティ手段を包含し、前記別のコードが、前記セキュリティ手段に対する複数の呼び出しインストラクションを組み込んでおり、各呼び出しインストラクションが、前記別のコード内における、実行可能コードのセットの必須ロケーションに含まれており、前記セキュリティ手段が、それぞれの呼び出しインストラクションによって呼び出された際に、改ざんを評価し、実行可能コードのセットを再形成し、前記各呼び出しインストラクションを前記再形成された実行可能コードのセットに置き換えるべく動作可能であるディジタル・データ保護装置。

【請求項２】

前記セキュリティ手段が、改ざんが検出されなかった場合に前記別のコードを削除するべく動作可能であるとする請求項１記載のディジタル・データ保護装置。

【請求項３】

前記セキュリティ手段が、前記必須ロケーションにおいてロケートされる暗号化されたコードを暗号解除し、かつ前記暗号化されたコードを対応する暗号解除されたコードに置き換えるべく動作可能であるとする請求項１～２のいずれかに記載のディジタル・データ保護装置。

【請求項４】
前記セキュリティ手段が、前記別のコード内に埋め込まれているものとする請求項１～３のいずれかに記載のディジタル・データ保護装置。
【請求項５】
前記セキュリティ手段が、前記別のコードによって使用されないロケーションに埋め込まれているものとする請求項４記載のディジタル・データ保護装置。
【請求項６】
前記実行可能コードが実行されたときに、少なくとも１つの埋め込みロケーションが識別され、前記埋め込みロケーションに、前記セキュリティ手段が書き込まれるものとする請求項５記載のディジタル・データ保護装置。
【請求項７】
前記別のコードをデコンパイルし、かつ前記デコンパイルされたコードを分析することによって１つの埋め込みロケーションが識別されるものとする請求項６記載のディジタル・データ保護装置。
【請求項８】
さらに、前記セキュリティ手段のロケーションを変更し、かつ前記呼び出しインストラクションを修正して新しいロケーションを参照させるべく動作可能な配置変更手段を包含するものとする請求項１～７のいずれかに記載のディジタル・データ保護装置。
【請求項９】
前記配置変更手段が、前記被保護データ内に含まれており、被保護コードが使用されている間にわたり反復的に動作するものとする請求項８記載のディジタル・データ保護装置。
【請求項１０】
前記実行可能コードが、前記被保護コードを生成するための実行可能インストラクションを包含するものとする請求項１～９のいずれかに記載のディジタル・データ保護装置。
【請求項１１】
前記実行可能コードが、暗号化された形式の前記被保護データを、暗号解除のための実行可能インストラクションとともに組み込んでいるものとする請求項１～１０のいずれかに記載のディジタル・データ保護装置。
【請求項１２】
前記実行可能コードの最初の実行が、実行のために前記暗号解除インストラクションをインストールし、続く前記暗号解除インストラクションの実行が、前記被保護データの暗号解除をもたらすものとする請求項１１記載のディジタル・データ保護装置。
【請求項１３】
前記暗号解除インストラクションが、当初は非実行可能形式でストアされており、かつ被保護ソフトウエアの実行を認可するために実行を必要とし、さらに前記ディジタル・データ保護装置が、前記コードのブロックを、少なくとも１つの変換鍵を必要とするアルゴリズムを使用して実行可能形式に変換するべく動作可能であり、かつ前記アルゴリズム内において使用するための変換鍵を、実行可能形式もしくは非実行可能形式のターゲットのコードのブロックを参照することによって導出するべく動作可能な変換手段を含み、それによって適切な変換鍵が、前記ターゲットのブロックが修正されていない場合に限り導出されるものとする請求項１２記載のディジタル・データ保護装置。
【請求項１４】
前記セキュリティ手段が、非実行可能形式でストアされる複数の実行可能コードのブロックであって、それぞれが前記被保護ソフトウエアの実行を認可するために実行を必要とする複数の実行可能コードのブロックを包含し、前記変換手段が、各ブロックを実行可能形式に変換するべく動作可能であるものとする請求項１３記載のディジタル・データ保護装置。
【請求項１５】
前記各ブロックの変換が、それぞれのターゲット・ブロックから導出される変換鍵によ

って達成されるものとする請求項１４記載のディジタル・データ保護装置。
【請求項１６】
少なくとも１つのブロックが、実行時に、続く実行のために別のブロックを実行可能形式に変換するべく動作可能であるものとする請求項１５記載のディジタル・データ保護装置。
【請求項１７】
各ブロックが、実行時に、続く実行のために別のブロックを実行可能形式に変換するべく動作可能であるものとする請求項１６記載のディジタル・データ保護装置。
【請求項１８】
前記ターゲット・ブロックもしくは各ターゲット・ブロックが、前記被保護ソフトウエア内に含まれているものとする請求項１３～１７のいずれかに記載のディジタル・データ保護装置。
【請求項１９】
前記ターゲット・ブロックもしくは各ターゲット・ブロックが、第１のセキュリティ手段内に含まれているものとする請求項１３～１７のいずれかに記載のディジタル・データ保護装置。
【請求項２０】
コードを変換するための前記アルゴリズムもしくは各アルゴリズムが、ＣＲＣアルゴリズムを含むものとする請求項１３～１９のいずれかに記載のディジタル・データ保護装置。
【請求項２１】
コードを実行するべく動作可能なプロセッシング手段、および前記実行可能コードがストアされるメモリ手段を備え、前記実行可能コードが、前記被保護データに関するスタート・ポイントとして前記ディジタル・データ保護装置内において示されるメモリ・ロケーションにスタート・ポイントを伴って前記メモリ手段内にストアされており、それによって前記プロセッサ手段が、前記被保護データのアクセスを求めるとき、前記実行可能コードの実行をもたらすものとする請求項１～２０のいずれかに記載のディジタル・データ保護装置。
【請求項２２】
前記実行可能コードが、実質的に暗号化されていない形式で前記被保護データを再形成するべく動作可能であるとする請求項１～２１のいずれかに記載のディジタル・データ保護装置。
【請求項２３】
前記被保護データが、少なくとも１つの実行可能インストラクションを含み、それが複数のステップを含み、前記ステップが、前記インストラクションを実施するために１を超える種類の順序で実行可能であり、前記実行可能コードが、前記実行可能コードの各実行時に変化する順序で前記ステップを生成することによって前記インストラクションを生成するべく動作可能であるものとする請求項１～２２のいずれかに記載のディジタル・データ保護装置。
【請求項２４】
前記ステップの順序が、各実行時に、実質的にランダムに選択されるものとする請求項２３記載のディジタル・データ保護装置。
【請求項２５】
前記ステップが、少なくとも１つのステップであって、前記被保護データの改ざんを検出するべく動作可能なセキュリティ手段の動作を開始するステップを含むものとする請求項２３または２４記載のディジタル・データ保護装置。
【請求項２６】
前記実行可能コードが、前記実行可能インストラクションが実行されることになるごとに、前記ステップを生成するべく実行可能であるものとする請求項２３～２５のいずれかに記載のディジタル・データ保護装置。

【請求項２７】
前記実行可能コードが、各実行時に前記被保護データの一部を使用可能な形式で提供し、かつ残りのデータを改ざんされた形式で提供するべく構成されており、それによって、前記被保護データのすべてを使用可能な形式で提供するためには１を超える数の実行が必要になるものとする請求項１～２６のいずれかに記載のディジタル・データ保護装置。
【請求項２８】
前記被保護データの各部分が、前記被保護データ内の完全な実行可能ルーチンに対応しており、それによって、前記被保護データを形成するルーチンの完全なセットが、前記実行可能コードの繰り返し実行によって生成可能であるものとする請求項２７記載のディジタル・データ保護装置。
【請求項２９】
前記実行可能コードの各実行が、先行して生成された使用可能コードの改ざんをもたらし、それによって、もっとも新しい実行によって生成されたコードのみが前記ディジタル・データ保護装置の動作の間において使用可能な形式になるものとする請求項２７または２８記載のディジタル・データ保護装置。
【請求項３０】
請求項１～２９のいずれかの請求項に従ったディジタル保護装置を含むメモリ手段を包含するコンピュータ・システム。
【請求項３１】
コンピュータ・システムにインストールされたとき、請求項１～２９のいずれかに従ったディジタル・データ保護装置として動作可能であるソフトウエアを含むデータ・キャリア。
【請求項３２】
コンピュータ・システムにインストールされたとき、請求項１～２９のいずれかに従ったディジタル・データ保護装置として動作可能であるコンピュータ・ソフトウエア。
【請求項３３】
コンピュータ・ソフトウエアの二次アイテムのための保護を提供するべく動作可能なコンピュータ・ソフトウエアであって、保護ソフトウエアが、１ないしは複数のセキュリティ・チェックに成功して完了したことに応答して被保護ソフトウエアの実行を認可するべく動作可能であり、かつ少なくとも１つの、非実行可能形式でストアされ、前記被保護ソフトウエアの実行を認可するために実行が必要となる実行可能コードのブロックを有するセキュリティ手段を包含し、さらに前記保護ソフトウエアが、前記コードのブロックを、少なくとも１つの変換鍵を必要とするアルゴリズムを用いて実行可能形式に変換するべく動作可能な変換手段を包含し、前記変換手段は、前記アルゴリズム内における使用のための変換鍵を、実行可能または非実行可能形式のターゲットのコードのブロックを参照することによって導出するべく動作可能であり、それによって適切な変換鍵が、前記ターゲットのブロックが修正されていない場合に限り導出されるものとするコンピュータ・ソフトウエア。
【請求項３４】
請求項３３に従ったコンピュータ・ソフトウエアを含むコンピュータ・メモリ・デバイス。
【請求項３５】
請求項３３または３４に従ったコンピュータ・ソフトウエアを用いて保護されるコンピュータ・ソフトウエアのアイテムを含むコンピュータ・システム。
【発明の詳細な説明】
【０００１】
本発明は、ディジタル・データのための保護構成に関する。
【０００２】
ソフトウエア・ファイルおよびデータ・ファイルを含むディジタル・データは、各種の理由から保護を必要とすることがある。たとえば、権限のないコピーに備えて保護が必要に

なることがある。ソフトウエアの場合は、ウィルス攻撃、ハッキングもしくは類似の行為による改ざんに備えて保護が必要とされることもある。

【０００３】

本発明は、ディジタル・データのための改良された保護構成を提供する。

【０００４】

本発明は、実行可能コードを包含するディジタル・データ保護構成を提供し、当該実行可能コードは、実行されたときに、被保護データを使用可能な形式で含む別のコードを生成することが可能になる、被保護データに関連する充分な情報を組み込んでいる。

【０００５】

この構成は、被保護データの改ざんを検出するべく動作可能なセキュリティ手段を包含することが可能であり、前述の別のコードは、セキュリティ手段を呼び出して任意の改ざんを評価する少なくとも１つのインストラクションを組み込んでいるものとすることができる。好ましくは、呼び出しインストラクションが、前述の別のコード内における、実行可能コードのセットの必須ロケーションに含まれているものとし、かつ、セキュリティ手段が、実行時に実行可能コードのセットを再形成し、呼び出しインストラクションに置き換えるべく動作可能であるとする。また好ましくは、セキュリティ手段が、改ざんが検出されなかった場合に前述の別のコードを削除するべく動作可能であるとする。

【０００６】

この構成は、さらに、セキュリティ手段のロケーションを変更し、かつ呼び出しインストラクションを修正して新しいロケーションを参照させるべく動作可能な配置変更手段を包含することができる。配置変更手段は、被保護データ内に含めることが可能であり、被保護コードが使用されている間にわたって反復的に動作することができる。

【０００７】

実行可能コードは、被保護コードを生成するための実行可能インストラクションを包含することができる。この実行可能コードの実行は、好ましくは実行のために暗号解除インストラクションをインストールし、続く暗号解除インストラクションの実行が、被保護データの暗号解除をもたらす。暗号解除インストラクションは、当初、非実行可能形式でストアされるが、その実行が、被保護ソフトウエアの実行を認可するために必要となるものとすることが可能であり、さらにこの構成が、前述のコードのブロックを、少なくとも１つの変換鍵を必要とするアルゴリズムを使用して実行可能形式に変換するべく動作可能であり、かつそのアルゴリズム内において使用するための変換鍵を、実行可能形式もしくは非実行可能形式のターゲットのコードのブロックを参照することによって導出するべく動作可能な変換手段を含み、それによって、ターゲットのブロックが修正されていない場合に限り適切な変換鍵が導出されるものとすることができる。

【０００８】

セキュリティ手段は、非実行可能形式でストアされる複数の実行可能コードのブロックであって、それぞれが被保護ソフトウエアの実行を認可するために実行を必要とする複数の実行可能コードのブロックを包含することが可能であり、変換手段は、各ブロックを実行可能形式に変換するべく動作可能であるとすることができる。各ブロックの変換は、好ましくはそれぞれのターゲット・ブロックから導出される変換鍵によって達成されるものとする。好ましくは少なくとも１つのブロックが、実行時に、続く実行のために別のブロックを実行可能形式に変換するべく動作可能であるとする。また好ましくは各ブロックが、実行時に、続く実行のために別のブロックを実行可能形式に変換するべく動作可能であるとする。

【０００９】

前述のターゲット・ブロックあるいは各ターゲット・ブロックは、被保護ソフトウエア内または最初のセキュリティ手段内に含まれているものとすることができる。

【００１０】

コードを変換するための前述のアルゴリズムあるいは各アルゴリズムは、ＣＲＣアルゴリズムを含むことができる。実行可能コードは、暗号化された形式の被保護データを、暗号

解除のための実行可能インストラクションとともに組み込むことができる。被保護データは、実行可能コードおよび／またはデータ・ファイルを包含することができる。この構成は、コードを実行するべく動作可能なプロセッシング手段、および実行可能コードがストアされるメモリ手段を備えることが可能であり、前述の実行可能コードは、被保護データに関するスタート・ポイントとしてこの構成内において示されるメモリ・ロケーションにスタート・ポイントを伴ってメモリ手段内にストアされており、それによってプロセッサ手段が被保護データのアクセスを求めるとき、実行可能コードの実行がもたらされるものとすることができる。実行可能コードは、好ましくは実質的に暗号化されていない形式で被保護データを再形成するべく動作可能であるとする。

【００１１】

被保護データは、少なくとも１つの実行可能インストラクションを含むことが可能であり、それが複数のステップを含み、それらのステップは、インストラクションを実施するために１を超える種類の順序で実行可能であり、実行可能コードは、当該実行可能コードの各実行時に変化する順序でステップを生成することによってこのインストラクションを生成するべく動作可能であるとすることができる。これらのステップの順序は、好ましくは各実行時に、実質的にランダムに選択される。これらのステップは、少なくとも１つのステップであって、被保護データの改ざんを検出するべく動作可能なセキュリティ手段の動作を開始するステップを含むことができる。実行可能コードは、実行可能インストラクションが実行されることになるごとに、ステップを生成するべく実行されるものとすることができる。

【００１２】

実行可能コードは、各実行時に被保護データの一部を使用可能な形式で提供し、かつ残りのデータを改ざんされた形式で提供するべく構成され、それによって被保護データのすべてを使用可能な形式で提供するためには１を超える数の実行が必要になるものとすることができる。各部分が、被保護データ内の完全な実行可能ルーチンに対応しており、それによって被保護データを形成するルーチンの完全なセットが、実行可能コードの繰り返し実行によって生成可能であるとすることができる。実行可能コードの各実行は、好ましくは先行して生成された使用可能コードの改ざんをもたらし、それによって、もっとも新しい実行によって生成されたコードのみがこの構成の動作間において使用可能な形式となる。

【００１３】

また本発明は、以上の定義のいずれかに従ったディジタル保護構成を含むメモリ手段を包含するコンピュータ・システムを提供する。

【００１４】

さらに本発明は、コンピュータ・システムにインストールされたとき、以上の定義のいずれかに従ったディジタル・データ保護構成として動作可能であるソフトウエアを含むデータ・キャリアを提供する。

【００１５】

さらに本発明は、コンピュータ・システムにインストールされたとき、以上の定義のいずれかに従ったディジタル・データ保護構成として動作可能であるコンピュータ・ソフトウエアを提供する。

【００１６】

さらにまた本発明は、コンピュータ・ソフトウエアの二次アイテムのための保護を提供するべく動作可能なコンピュータ・ソフトウエアを提供し、この保護ソフトウエアは、１ないしは複数のセキュリティ・チェックに成功して完了したことに応答して被保護ソフトウエアの実行を認可するべく動作可能であり、かつ少なくとも１つの、非実行可能形式でストアされ、被保護ソフトウエアの実行を認可するために実行が必要となる実行可能コードのブロックを有するセキュリティ手段を包含し、さらにこの保護ソフトウエアは、前述のコードのブロックを、少なくとも１つの変換鍵を必要とするアルゴリズムを用いて実行可能形式に変換するべく動作可能な変換手段を包含し、当該変換手段は、さらに、このアルゴリズム内における使用のための変換鍵を、実行可能または非実行可能形式のターゲット

のコードのブロックを参照することによって導出するべく動作可能であり、それによって、ターゲットのブロックが修正されていない場合に限り適切な変換鍵が導出されるものとなる。

【００１７】

さらにまた本発明は、上記のコンピュータ・ソフトウエアを含むコンピュータ・メモリ・デバイスを提供する。

【００１８】

さらにまた本発明は、上記のコンピュータ・ソフトウエアを用いて保護されるコンピュータ・ソフトウエアのアイテムを含むコンピュータ・システムを提供する。

【００１９】

以下、添付図面を参照し、例示のみを目的として本発明の例についてより詳細に説明する。

【００２０】

本発明を説明するために、まず、本発明を実施することができる単純なコンピューティング装置について簡単に述べておくことは有用である。図１は、プロセッサ１２を含む従来のハードウエア・コンポーネントを備えたコンピューティング装置１０を示しており、そこにはキーボード１４およびディスプレイ１６等の適切な入力および出力デバイスが接続されている。メモリは、ソフトウエアの実行間におけるプロセッサ１２による使用のために、ハード・ドライブ１８あるいはそれに類似の大容量メモリ・デバイスの形式、およびＲＡＭ（ランダム・アクセス・メモリ）２０の形式で備えられている。この装置は、ＩＢＭ　ＰＣとして知られるタイプ、あるいはそれに等価もしくは類似のマシンとすることができる。

【００２１】

図２は、装置１０のドライブ１８上にストアされたソフトウエアを実行するために、従来はＲＡＭ２０がどのように使用されていたかについて示している。最初にソフトウエアは、プロセッサ１２によってドライブ１８からＲＡＭ２０内にロードされ、その結果、マシン・コードのステップ２２（「ステップ１」、「ステップ２」等として図示）のシーケンスが、ＲＡＭ２０内に順次ストアされる。このステップのシーケンスは、「終了」インストラクション２４を伴って終了する。ここで認識されようが、実際の例においては、ステップの数が図示のそれと同じでなくてよく、通常は非常に大きな数となろう。

【００２２】

最初のステップ「ステップ１」には、習慣的にコードの「ローダ」ブロック２６が前置され、実行時にはそれが、入力／出力デバイス１４、１６用のドライバ、あるいはそのほかの各種のアプリケーション・プログラムによって広く共有されるソフトウエア・リソースといったほかのリソースの識別をもたらし、ＲＡＭ内のソフトウエアによる使用を可能にする。

【００２３】

図２に示されている形式にソフトウエアをロードする間に、装置１０のプロセッサ１２は、図示の実行可能データのブロックに関するスタート・ポイントを、つまり「ローダ」２６が始まるＲＡＭ２０内のメモリ・ロケーションを記録する。図２においては（およびそのほかの図面において）、このスタート・ポイントの表記が、矢印２８の使用によって表されている。

【００２４】

装置１０に、この時点においてＲＡＭ２０内に収められているソフトウエアの実行が指示されると、プロセッサ１２は、まずスタート・ポイント２８として識別されるＲＡＭ２０のロケーションに行き、そのロケーションにおいて見つかった実行可能データの実行を開始する。つまりプロセッサ１２は、まず「ローダ」２６を実行し、それに続いてステップ２２を実行し、最終的に「終了」ステップ２４において終了する。

【００２５】

図２から明らかになることは、ＲＡＭ２０内のソフトウエアが暗号化されていない形式で

さらされていることであり、したがってウィルス・ソフトウエアによる攻撃、権限のないコピー、ウィルスの生成その他によって当該ソフトウエアの攻撃を謀る悪意のあるユーザによる分析に対して無防備であるということである。

【００２６】

（例１）

図３は、本発明に従った基本構成を示しており、それによってソフトウエアの保護が改善される。図３Ａは、本発明に従って保護されているソフトウエアが、最初にドライブ１８からロードされたときのＲＡＭ２０の状態を示している。

【００２７】

図３Ａの状態においては、ＲＡＭ２０が、ここでは「エンジン」３０と呼んでいる、スタート・ポイント２８にロケートされた実行可能コードの先頭ブロックを含んでおり、それが被保護ソフトウエアであることがプロセッサ１２に知らされる。ＲＡＭ２０の、「エンジン」３０の下側３２は空である。

【００２８】

「エンジン」３０は、上に述べたように実行可能コードである。しかしながら「エンジン」３０の実行可能は、被保護ソフトウエアを実行するための実行可能ではない。むしろエンジン３０は、「エンジン」３０が実行されたときに被保護ソフトウエアを再形成し、それをＲＡＭ２０の空領域３２に書き込むために実行可能である。つまり、単純な態様においては「エンジン」３０を、被保護データの暗号化されたバージョン（または暗号化されたバージョンが入手できるアドレス）、およびソフトウエアを暗号解除し、それを領域３２に書き込むための実行可能ソフトウエアから構成することができる。

【００２９】

この書類内において、暗号化という用語およびそれに関連する用語は、データがその使用可能な形式から直ちに実行可能でない別の型式に変換される、圧縮を含むあらゆるテクニックを指していることを明確に理解される必要がある。暗号解除ならびにそれに関連する用語は、伸張プロセスを含め、その逆のプロセスを指すために使用されている。

【００３０】

図３Ａから気付くことは、エンジン３０がスタート・ポイント２８に位置していることである。その結果、被保護ソフトウエアが呼び出されたとき、プロセッサ１２は、最初にロケーション２８に行き、したがって「エンジン」３０を構成するコードを実行することになる。つまり最初に、図３Ｂに示されているように、被保護ソフトウエアが非暗号化形式で再形成されてスペース３２が埋められる。

【００３１】

プロセッサ１２が「エンジン」３０の実行を続けると、その結果として図３Ｂの３４に示されているようにプログラム・ポインタが「ステップ１」の先頭に到達するが、その時点までには、「ステップ１」および残りの被保護データがスペース３２内に再形成されており、それにより被保護ソフトウエアの実行が開始されることになる。

【００３２】

ここで理解されるであろうが、「エンジン」３０の実行は、図２に示されている従来の構成に比較すると、追加となるステップを構成する。しかしながら、マシン・コードにおける「エンジン」３０の動作が充分に迅速となり、この余剰なステップをユーザに意識させないであろうと認識される。

【００３３】

図３に示されているような基本動作について説明したが、この被保護データが、図２の従来の構成に比較して、コピーおよび分析に対してより脆弱でないことを理解することができる。これは、被保護データが、「エンジン」３０を実行してしまうまで、つまり被保護ソフトウエアが呼び出されてしまうまで、攻撃、コピー、または分析に使用できないことによる。被保護データをドライブ１８からＲＡＭ２０にロードしただけでは、攻撃、コピー、または分析の前に被保護データをさらすという意味では充分でない。

【００３４】

図３Ａおよび３Ｂの構成は、図２の構成に対して改善されているが、「エンジン」３０を実行してしまった後の被保護データが、図３Ｂにおいても図２の形式のＲＡＭ内と同じ態様で暴露されることは明らかである。さらに保護を向上するために、次に示す方法に従ってスペース３２内にデータを再形成するべく「エンジン」３０を修正することができる。

【００３５】

（例２）

図３Ｃ～３Ｅの例においては、ＲＡＭ２０が、当初、図３Ａに示されている場合と同様に、この例では３０Ａというラベルの付された「エンジン」および空のＲＡＭ＿３２だけを有している。「エンジン」３０Ａの実行は、この場合にもスペース３２を充填することになる。しかしながら、図３Ｄと図３Ｂを比較することによって、２つの変更が明らかになる。まず、「ステップ２」が「呼び出し」インストラクション３６に置き換えられ、「エンジン」３０Ａによって「終了」インストラクション２４の後に「保護」ブロック３８が挿入されている。これは、以下の方法に従って達成される。

【００３６】

「エンジン」３０Ａは実行中に被保護データを暗号解除する一方、「エンジン」３０Ａは、結果として得られる暗号解除済みコードをモニタし、コードの特定のパターンまたはシーケンスを探す。その詳細については、例３との関係から説明を後述する。「エンジン」３０Ａは、１つのシーケンスまたは１を超える数のシーケンスを探すことができる。当然のことながら、探しているシーケンスの長さは、そのシーケンスがロケートされる尤度(likelihood)に影響を与える。この例においては、「エンジン」３０Ａが「ステップ２」に対応するシーケンスを探す。このあらかじめ決定済みのコードのセット、すなわちこの場合であれば「ステップ２」がロケートされると、「エンジン」３０Ａは、その、あらかじめ決定済みのコードのセットを書き込むことなく、それに代えて、「保護」ブロック３８の呼び出しを指示する「呼び出し」インストラクション３６を書き込む。つまり、「エンジン」３０Ａの実行が完了したときには「ステップ２」に代わって「呼び出し」インストラクション３６が存在し、被保護データが完全に可視にはならない。

【００３７】

被保護ソフトウエアの実行時においては、いずれかの時点で「ステップ２」の開始が期待されるロケーションにプログラムが到達し、「呼び出し」インストラクション３６を見ることになる。それにより「保護」ブロック３８の実行が開始する。ブロック３８は、２つの機能を有する。第１においては、「ステップ２」用のコードが（エンジン３０の動作に類似する方法に従って）生成され、「呼び出し」インストラクション３６に上書きされ、それによって被保護コードの暗号解除が完成する。ブロック３８は、さらに従来のウィルス保護またはそれに類似する検査等のセキュリティ・チェックを行い、ウィルス保護が適正に実行されていること、および改ざんが検出されていないことを保証する。不適切なことがまったくない場合には、保護ブロック３８が終了し、保護ブロック３８に対する呼び出しが発生したロケーションに実行を返す。そのロケーションは、この時点において「ステップ２」用のコードの開始になっている。その後、通常の方法に従って「ステップ２」からオペレーションが継続する。

【００３８】

この場合にも、マシン・コードにおける完全な動作によって、この追加のステップを、ユーザが気付くことのできない充分な速さにすることが可能である。

【００３９】

このように、被保護コードは、そのコードが実行されるまで完全に暗号解除されず、そのためウィルス攻撃、権限のないコピー、あるいは悪意のある分析のリスクが著しく抑えられることが理解できる。

【００４０】

「保護」ブロック３８がチェック時に何らかの不適切なことを発見した場合には、適切な報復が実施される。単純な例においては、特にウィルス攻撃に対して効果的であるが、ブロック３８がＲＡＭ２０内に収められている被保護データのバージョン（つまり、ＲＡＭ

内のセクション３２の全内容）を削除し、それによってウィルス、または被保護コードを汚染したウィルスの効果を削除する。

【００４１】

ここで明らかになるであろうが、マシン・コードにおいて、また非常に多くの行のコードを伴う現実的な状況において動作している場合には、多くの「呼び出し」インストラクション３６が「エンジン」３０Ａによって作成され、悪意のあるウォッチャがそのロケーションを予測し、あるいはマシン・コードの複雑なブロック内に見つかる従来的な呼び出しインストラクションと区別することが困難になる。したがって、悪意のあるウォッチャが保護の動作を分析すること、あるいはウィルスが保護を無効化することは、はるかに困難なものとなる。この保護は、エンジン３０Ａに、それ自体が変化する、おそらくは「呼び出し」インストラクションが挿入されるごとに変化するコードのブロックを待ち構えさせ、外部のオブザーバに対して呼び出しインストラクションの挿入を擬似ランダムにすることによって、さらに強化することができる。

【００４２】

（例３）

図４Ａおよび４Ｂは、ある意味で図３Ｃ、Ｄ、およびＥと類似した保護構成を示しているが、以下のような追加のセキュリティを提供する。

【００４３】

例２の構成は、保護されたブロックの末尾に配置される単独に識別可能なブロック３８を使用するが、この例は、ブロック３８と類似に動作するが保護されているソフトウエア内に埋め込まれるブロックを使用する。この埋め込みプロセスは、まず、図４Ａを考察することによって説明が可能である。図４Ａは、エンジン（この場合には３０Ｂの符号を付している）が被保護ソフトウエアの多数行のコードを生成したが、実行をまだ完了していない状態であり、かつブロック３８に対応する保護を生成していない状態のＲＡＭ２０を示している。図４Ａも略図的ではあるが、図３Ｃ～３Ｅに示されているよりは詳細にＲＡＭ２０の状態を示している。詳細に述べれば、ＲＡＭ２０の左側に付されている番号は、対応するマシン・コードのリスティングの行番号を示しており、マシン・コードによって提供される各種のプロシージャの間の境界を表すが、これらの境界は、ＲＡＭ２０を区切るラインによっても表されている。つまり、エンジン３０Ｂの下側に、最初のプロセス「プロセス１」用のコードが第１，０００行まで延び、それに続いて第２のプロセス「プロセス２」用のコードが第１，５００行まで延び、さらに第３のプロセス「プロセス３」用のコードが第２，５００行まで、最後に第４のプロセス「プロセス４」用のコードが第３，２００行まで延びている。

【００４４】

エンジン３０Ｂは、実行を完了するために、ＲＡＭ２０内のエンジン３０Ｂから第３，２００行のリストの末尾までの間にブロック３８に対応するインストラクションの挿入を試みる。これは、まずＲＡＭ２０の内容を調べて、データを含んでいない領域を識別することによって達成される。プロセス「プロセス１」は、ＲＡＭ２０内において連続して示されているが、それにもかかわらず、各種の理由からデータを含まないメモリの領域が存在する。詳細に述べれば、アッセンブラおよびコンパイラ・ソフトウエアは、一般にそのルーチンに非効率な部分を有し、その結果としてＲＡＭにデータを持たないデータレス・エリアが作られる。エンジン３０Ｂは、これらのエリアの識別を試みる。しかしながら、エンジン３０Ｂが機能するアルゴリズムは、ほかの理由からデータレス・エリアが生じる可能性があることも考慮しなければならない。たとえば、プロセス「ステップ１」等が、変数のカレント値、あるいは実行間にセットされたフラグの状態を記録するためにスクラッチ・パッドにアクセスを必要とすることがある。ＲＡＭのこれらのエリアは、最初は、ゼロを含むべくセットされることがあり、データを有していないように現れる。

【００４５】

つまりエンジン３０Ｂは、まずＲＡＭ２０内を調べて、データレスであるように見えるあらゆるエリアを識別する。見かけ上のデータレス・エリアがロケートされるごとに、その

ロケーションがエンジン３０Ｂによって、将来の参照のために記録される。図４Ａにおいては、たとえばプロセス「プロセス２」内に比較的大きなそれらしいエリアがあり、ゼロのみを含んでいることから見かけ上はデータレスである。プロセス「プロセス３」内においては、より小さいゼロを含んでいるエリアが明らかになる。実際例においては、さらに多くの、見かけ上のデータレス・エリアがロケートされる可能性がある。

【００４６】

すべてのデータレス・エリアが識別された後は、エンジン３０Ｂが、それらを分析していずれが真のデータレスであり、いずれがそのように見えるだけであるかを決定することができる。たとえば、エンジン３０Ｂは、ソフトウエアの全体を逆アッセンブルし、逆アッセンブルしたリスティングを評価して見かけ上のデータレスとして識別されたエリアを呼び出すインストラクション、もしくはそれらのエリアにジャンプするインストラクション、あるいはそれらのエリアに対して書き込みまたは読み出しを行うインストラクションを探す。その種のインストラクションが、それらのエリアの識別を行っている場合には、対応するエリアがソフトウエアによって、たとえばスクラッチ・パッドとして使用される可能性が高い。したがって、そのエリアは、ソフトウエアの実行間に少なくとも断続的にデータ用に使用されることになるため、そのエリアを見かけ上のデータレス・エリアのリストから削除する。

【００４７】

この方法に従って、見かけ上のデータレス・エリアのそれぞれのチェックを完了した後は、エンジン３０Ｂがデータレス・エリアの使用を開始することができる。この例においては、プロセス「プロセス３」内の２つのエリアが、そこを識別する呼び出しまたはジャンプ・インストラクションを有していることがわかり、したがって、データレス・エリアのリストから削除され、プロセス「プロセス２」内のエリアがリスト内に残されているものと仮定する。

【００４８】

プロセス「プロセス２」内のデータレス・エリアは、エンジン３０Ｂによって、ブロック３８の機能の１ないしは複数、たとえばセキュリティ・チェック等の機能が挿入され、また暗号解除アルゴリズムを表す実行可能コードによって充填される。この充填は、図４Ｂのハッチングによって示されているように、第１，３００行から開始する。

【００４９】

このとき第１，３００行に生成されたコードのブロックを使用するために、続いてエンジン３０Ｂは、ソフトウエア内のいずれかの場所、たとえばプロセス１内の第８００行にあるデータのブロックを修正する。まず、第８００行において開始するセクションの内容が、エンジン３０Ｂによって読み出される。これには、そのエリアのオリジナルの内容の暗号化に使用される暗号化アルゴリズムが含まれる。その後、暗号化済みバージョンが、第８００行から開始するエリアに書き戻される。図４Ｂにおいては、これが暗号化済みデータＥのセットとして示されている。この暗号化済みデータには、エンジン３０Ｂによって挿入される呼び出しインストラクションＣが先行する。

【００５０】

呼び出しインストラクションＣは、第１，３００行のコードを呼び出す。その結果、プロセス「プロセス１」の実行が第８００行に到達すると、第１，３００行が呼び出され、セキュリティ機能が実行された後に、第１，３００行のエリアの暗号解除ルーチンが実行される。これらの暗号解除ルーチンは、その呼び出しが行われたエリアに作用するべく記述されており、それによってエンジン３０Ｂの動作を反転して第８００行のエリアを暗号解除した後、プロセス「プロセス１」の暗号解除済みの部分を第８００行のエリアに書き戻す。

【００５１】

第１，３００行のエリアからの暗号解除ルーチンが実行された後は、コントロールが呼び出しのソース（第８００行）に戻り、暗号解除済みのコードの実行が可能になる。

【００５２】

「エンジン」３０Ｂが、その機能を埋め込むための充分なデータレス・エリアをソフトウエア内に見つけられなかった場合には、追加の機能が第３，２００行に書き込まれる。

【００５３】

この例は、次に示す表を考察することによってさらに詳細に説明することが可能であり、この表の左側の列は行番号を、中央の列は１６進数表示のマシン・コード・バイトを、右側の列は逆アッセンブル後のインストラクションをそれぞれ表している。

【００５４】

【表１】

:01007B45	55	push ebp
:01007B46	8BEC	mov ebp, esp
:01007B48	51	push ecx
:01007B49	53	push ebx
:01007B4A	56	push esi
:01007B4B	33F6	xor esi, esi
:01007B4D	57	push edi
:01007B4E	56	push esi
:01007B4F	8975FC	mov dword ptr [ebp-04], esi
:01007B52	E806F8FFFF	call 0100735D
:01007B57	8975FC	mov dword ptr [ebp-04], esi
:01007B5A	E853F8FFFF	call 010073B2
:01007B5F	8945FC	mov dword ptr [ebp-04], eax
:01007B62	FF30	push dword ptr [eax]
:01007B64	E8E0F7FFFF	call 01007349

【００５５】

表１を参照すると、複雑なソフトウエア内には頻繁に現れるように、短いセクションのリストの中に３つの呼び出しインストラクションが含まれていることがわかる。各呼び出しインストラクションは、２バイトのＥ８およびそれに続く、呼び出し先アドレスを与える４バイトからなる。

【００５６】

このように表１は、プロセス「プロセス１」内における第８００行から開始するセクション等の、被保護ソフトウエアの実際のリスティングを表している。

【００５７】

前述したようにエンジン３０Ｂが、たとえば第０１００７Ｂ４５行に呼び出しインストラクションを挿入した後は次のようになる。

【００５８】

【表２】

:01007B45	E8XXXXXXXX	call xx xx xx xx
:01007B4A	56	push esi
:01007B4B	33F6	xor esi, esi
:01007B4D	57	push edi
:01007B4E	56	push esi
:01007B4F	8975FC	mov dword ptr [ebp-04], esi
:01007B52	E806F8FFFF	call 0100735D
:01007B57	8975FC	mov dword ptr [ebp-04], esi
:01007B5A	E853F8FFFF	call 010073B2
:01007B5F	8945FC	mov dword ptr [ebp-04], eax
:01007B62	FF30	push dword ptr [eax]
:01007B64	E8E0F7FFFF	call 01007349

【００５９】
表１と表２を並べて考察すると、表２の最初の行に挿入された呼び出しインストラクションが、特にほかと異なる形には見えないことが明らかになる。つまり、これらのセキュリティ機能をディセーブル(disable)するためにセキュリティ機能に対する呼び出しを識別することは、実際上、達成可能となりにくい法外に複雑なタスクになる。たとえば、潜在的なハッカーが、ソフトウエア内にセキュリティ機能が埋め込まれていることを理解していた場合であっても、セキュリティ機能をディセーブルするためには、ソフトウエア内のすべての呼び出しインストラクションを方法論的に調べて、セキュリティ機能を呼び出している呼び出しインストラクションがあるか否か、また、いずれがそれを行っているかということ明らかにする必要がある。今日の複雑なソフトウエア・アプリケーションのマシン・コード・リスティングは、非常に多くの呼び出しインストラクションを含んでいる。
【００６０】
上記の説明から明らかになろうが、表２は、呼び出しインストラクション（図４Ｂの第８００行のＣ）が挿入されたが、続くコードのセクションがまだ暗号化されていない中間段階を示している。
【００６１】
また、数行の正当なコードが新しい呼び出しインストラクションによって上書きされていることも理解できる。これらの行を、セキュリティ機能の実行時に再形成されるようにセキュリティ機能内に記録することも可能であり、またそれに代えて、特定のセキュリティ機能に対する呼び出しインストラクションが、特定ターゲット行のコードに代えてだけ使用され、その結果、特定のセキュリティ機能が呼び出されるごとに、同一のコードが要求されてその呼び出しインストラクションが置き換えられるといったことも可能である。
【００６２】
さらに、呼び出しインストラクションが、ソフトウエア内のいずれかの場所からの正当な呼び出し先となるコード・ブロックを上書きしないようにする必要があり、それを行うとソフトウエアがフェイルすることも明らかになろう。したがってエンジン３０Ｂは、呼び出しインストラクションを書き込むことが可能なロケーションを探し出し、見かけ上のデータレス・エリアの識別に関連して前述した方法に従って、その種のロケーションを識別

するべく構成されている。

【００６３】

（例４）

図５Ａおよび５Ｂは、もう１つの、例２の構成に類似した保護構成を示している。詳細に述べれば、これにおいても保護ブロック４０が使用されている。

【００６４】

図５Ａは、最初に被保護コードがドライブ１８からＲＡＭ２０にロードされたときの状態を示している。この例における被保護コードは、プロセッサ１２によって４２にロードされるプログラムであり、そのスタート・ポイントが２８により示されている。この例においては、例証のみを目的として、スタート・ポイントがメモリ・ロケーション１０，０００として示されている。

【００６５】

プログラム４２のインストールに加えて、ドライブ１８からのダウンロードのオペレーションは、図５Ａに示されているように、メモリ・ロケーション１２，０００から開始する保護ブロック４０もインストールする。この保護ブロック４０は、後述する理由から「エンジン」４４を組み込んでいる。ブロック４０に対する呼び出しは、例２との関連から説明を前述したプロセスに類似の方法に従ってプログラム４２内に書き込まれる。

【００６６】

保護ブロック４０は、図３Ｄのブロック３８に類似した態様で動作する。言い換えるとプログラム４２は、改ざんの評価のためにブロック４０にジャンプする。

【００６７】

この種の性質を持った保護ジャンプは、呼び出しを待ち構えて、何らかの形式の保護コードと思われるコードを含むメモリ・ロケーションに規則的にジャンプする呼び出しを識別し、上書きによって呼び出しインストラクションをディセーブルする悪意のあるウォッチャもしくはウィルスによる攻撃ならびに無効化の攻撃を受けやすいと考えられる。

【００６８】

この例においては、このタイプの攻撃が次のようにして妨げられる。保護ブロック４０に対する最初の呼び出しインストラクションが現れたとき、保護ブロックが実行されて改ざんの評価が行われ、それが成功して完了すると、エンジン４４が実行される。「エンジン」４４は、２つの機能を実行する。一方の機能においてエンジン４４は、プログラム４２内の呼び出しインストラクションを異なるメモリ・ロケーション、たとえば１３，０００等を参照するように書き直す。これを行う前に「エンジン」４４は、ロケーション１３，０００に、ブロック４０を収める充分な空のメモリが存在することを保証するプロシージャを実行している。エンジン４４の他方の機能は、メモリ・ロケーション１３，０００から開始する保護ブロック４０（および４４自体）を再形成することである。したがって、保護ブロック４０が移動され、メモリ・ロケーション１２，０００に対する呼び出しインストラクションが、ロケーション１３，０００を呼び出すインストラクションによって上書きされる。その結果として、セキュリティ機能に対する呼び出しインストラクションが、プログラム４２の実行の間に絶えず変更され、それにより悪意のあるユーザまたはウィルスが保護ブロック４０に対する呼び出しを認識し、ディセーブルする上での困難が大幅に増加する。

【００６９】

たとえば、あるハッカーが、そのロケーションに対する呼び出しを分析することによって、セキュリティ・コードのロケーションを識別したと確信している場合に、そのハッカーは、そのロケーションに対するすべての呼び出しを識別してそれらをディセーブルする単純なコードを記述すると考えられる。しかしながら、それぞれが異なる場所にセキュリティ・ブロックを有することになるため、そのソリューションは、特定のソフトウエアのすべてのコピーに対して包括的に適用できない。

【００７０】

配置変更された保護ブロック４０を図５Ｂに示す。

【００７１】
（例５）
図６Ａおよび６Ｂは、さらに別の例のデータ保護を示している。この例においては「エンジン」５０が使用されて、例１および図３Ａおよび３Ｂに関連して前述した説明に類似の方法に従って空のＲＡＭセクション５２内にコードのブロックが作られる。つまり、被保護コードが最初にドライブ１８からＲＡＭ２０にロードされるときは、エンジン５０がＲＡＭ内にインストールされ、その残りが５２において空となっている図６Ａの左側に示されたポジションの関係になる。この例においては、エンジン５０が、サブルーチンもしくは他のメイン・プログラム内の従属的なオペレーションのスタート・ポイントに配置される。そのサブルーチンが呼び出されると、「エンジン」５０が実行されて領域５２を充填するステップが生成される。それにより図６Ａの右側に示されているように、４つのステップ「ステップ１」、「ステップ２」等を領域５２内に番号順に伴うポジションの関係になる。「エンジン」５０が実行されると、続いてステップ（「ステップ１」等）が実行され、サブルーチンが実行される。
【００７２】
図６Ｂは、このサブルーチンが次に呼び出されたときの状態を示している。最初は、図６Ｂの左側に示されているように、ＲＡＭ２０が、このサブルーチンの最初の実行が終了したときの状態、すなわち図６Ａの右側に示されている状態になっている。詳細に述べれば、「エンジン」５０が、まだコードのブロックのスタート・ポイントに存在している。つまり、このサブルーチンが次に呼び出されたとき、「エンジン」５０が再度実行される。しかしながら「エンジン」５０は、実行ごとに異なる順序に従って、好ましくはランダムもしくは擬似ランダムの順序に従ってステップを生成するべく記述されている。これは、サブルーチンの実行に不利な影響をもたらすことなく達成することが可能であり、実際、「プリント」ステートメント等の多くの従来のルーチンは、マシン・コードで記述されたときに数千行の長さになることもあり、実質的にその任意の行は、実質的に任意の順序に従って実行可能である。特定の順序に従った記述を必要とする行のグループがある場合には、それらの行をブロックとして扱い、単一のステップとして記述すればよく、その結果、そのブロック内の順序は保存されるが、そのブロックのポジジョンは、「エンジン」５０の実行ごとに変化する。
【００７３】
１ないしは複数の「ステップ」を記述し、ウィルス攻撃または改ざんに抗する保護ルーチンを実施し、あるいは呼び出すことができる。それを行う場合には、その「ステップ」がサブルーチンの実行ごとに異なるポジションに現れることになる。したがって悪意のあるウォッチャは、絶えず変化するコードを見ることになり、ディセーブルのためにそれを分析して保護をロケートすることが困難もしくは不可能になる。それとは別に、悪意のあるウォッチャが、あるステップを保護に関連するものとして認識し、たとえばそのステップを上書きすることによってそれをディセーブルした場合にも、この悪意のある介入自体が、次にこのサブルーチンが呼び出された時点において「エンジン」５０のオペレーションによって上書きされ、保護の動作が復元される。
【００７４】
（例６）
ＲＡＭ２０が図３Ｂもしくは図６Ａおよび６Ｂの右側に示されている状態にあるときは、完全な被保護データが暗号解除された形式で可視になる。それにもかかわらず前述した保護の形式がコードを保護することは認識されているが、同時に一部の状況においては、暗号化されていない完全なコードの暴露が、それがいかに一時的な暴露であったとしても保護の弱点を表すことが認識されている。さらに別の例は、どのようにすればこの困難が克服できるかについて例証する。再度、図３Ａに示されている状態のＲＡＭ２０を伴う図７Ａから開始するが、それにおいては「エンジン」６０がサブルーチンのスタート・ポイントにあり、それを除くＲＡＭ２０は、６２において空になっている。
【００７５】

このサブルーチンが呼び出されると、プロセッサ１２がスタート・ポイント２８に行き、「エンジン」６０の実行を開始する。「エンジン」６０の最初の実行時に、スペース６２内にコードのブロックが３つ生成される（この例においては、本発明の原理を例証するために３という数が選択されているが、これが本発明の範囲を限定することはない）。これらのブロックを図７Ｂに示す。つまり図７Ｂにおいては、６４に「ステップ１」が作成され、その後にアスタリスクによって表されている、改ざんされているか、あるいは意味のないデータの２つのブロック６６が続いている。それに代えて、ブロック６６を空のままにしておくことも可能であるが、改ざんされているか、あるいは偽のデータを生成することは、より大量のコードを提供することによって、悪意のあるウォッチャもしくはウィルスによる分析をより困難にすると認められる。

【００７６】

「エンジン」６０によって生成されたときの「ステップ１」は、実行を「エンジン」６０の先頭に返す「リターン」ステートメント６８を伴って完了する。つまり、「ステップ１」の実行完了後には、「エンジン」６０が再度実行される。「エンジン」６０は、レジスタをセットして最初の実行がすでに生じたことを記録する。つまり、第２の実行が進行中であるとの認識の下に、「エンジン」６０は、ブロック７０の前後の改ざんされたブロック６６とともに、その７０に「ステップ２」を生成する。

【００７７】

この場合においても、ブロック６６を空のままとしてもよい。

【００７８】

この時点において「ステップ２」の実行が可能になり、その後には、７２の「エンジン」６０へのリターンが続く。その後「エンジン」６０の３番目の実行が、２つの改ざんされたブロック６６とともに「ステップ３」を７４に生成し、「ステップ３」の実行を可能にする。

【００７９】

つまりこの方法に従って「エンジン」６０の動作を繰り返すことによって、プロセッサ１２が「ステップ１」；「ステップ２」；「ステップ３」の完全なシーケンスを実行することが可能になるが、いずれの時点においてもＲＡＭ２０から完全なシーケンスが可視になることはなく、したがってこのシーケンスの分析もしくは改ざん、あるいはそれ自体もしくはそれが含む保護のディセーブルを困難な、または不可能なものとする。これにおいても悪意のあるウォッチャは、絶えず変化する（または「突然変異する」）コードを見ることになり、したがって適切に分析して攻撃することが実質的に不可能になる。

【００８０】

ここで、デバッガ・ソフトウエアを使用するハッカーに対する保護に特に適している、さらに別の例を示すことができる。ハッカーは、ソフトウエアのセキュリティを攻撃するためにデバッガ・ソフトウエアをよく使用する。デバッガ・プログラムは、基本的に、複雑なソフトウエア内の誤りをチェックする正当な運用を意図している。デバッガ・プログラムは、一般に、ソフトウエア内の、オペレータが検討を希望するポイントに割り込みコマンドを挿入することによって機能する。ソフトウエアの実行が割り込みコマンドに到達すると、コントロールがデバッガ・プログラムに渡され、続いてそれが次のコードのブロックを調べて、ハイ・レベルまたはロー・レベルの言語を用いて結果をオペレータに提供し、それによってオペレータは、メイン・プログラムの実行の継続が許可される前にそのブロックをチェックすることが可能になる。

【００８１】

セキュリティ・ルーチンを迂回しようとするハッカーによって使用される１つのアプローチは、デバッガ・プログラムを使用してセキュリティ・プロシージャが呼び出されるメイン・プログラム内のポイントを探すことである。そのポイントには、メイン・プログラムの実行時にそれらのルーチンがジャンプされるように、ジャンプ・コマンドが挿入される。より今日的なソフトウエアは、充分に複雑であり、メイン・プログラム内のこれらのポイントをロケートするタスクが困難になっているが、それにもかかわらず熟練した根気強

いハッカーが、最終的にそれらを見つけ出せる可能性はある。一度それらを見つけ出した後のハッカーは、セキュリティ・ルーチンをディセーブルすることが可能であり、もっとも複雑なルーチンさえもそれによって無効化される。

【００８２】

（例７）

図８は、本発明を実施するさらに別の例を示している。図８には、２つのソフトウエアのブロック１２０、１２２を含むＲＡＭ２０が示されている。最初のブロック１２０は、本発明の構成によって保護されているソフトウエアである。ブロック１２２は、保護構成を実施するセキュリティ・ブロックである。被保護ソフトウエア２０は、セキュリティ・ヘッダ１２６が前置されたアプリケーション１２４（ワード・プロセッサ・アプリケーション等）を含んでいる。

【００８３】

アプリケーション１２４は、暗号化された形式でストアされることになり、圧縮を伴うこともある。つまりアプリケーション１２４は、これまでの提案に従いセキュリティ・ヘッダ１２６によって提供されるコントロールの下に実行可能形式に変換されるまで、すなわちコンピュータ・システム内に保持されているソフトウエア・ライセンスの詳細、ユーザの詳細、および現在日時（ソフトウエアが指定の期間に限ってライセンスされる場合）、その他のチェックといった各種のセキュリティ・チェックが行われた後にアプリケーション１２４が暗号解除され、伸張されるまで使用することができない。これらのチェックが成功して完了すると、ヘッダ１２６がアプリケーション１２４の暗号解除および伸張を行い、プロセッサのプログラム・ポインタをアプリケーション１２４の先頭に移動し、それによってコントロールをアプリケーション１２４に渡す。このタイプの保護は、多くの状況において有効となり得るが、前述したように「ジャンプ」コマンドを挿入してセキュリティ・ヘッダ１２６によって行われる１ないしは複数のチェックを迂回し、それによってすべての正常なセキュリティ・チェックが行われることなくアプリケーション１２４の実行を可能にするハッカーの攻撃を受けやすい。

【００８４】

セキュリティ・ブロック１２２は、この欠点を解決するために提供されている。

【００８５】

セキュリティ・ブロック１２２は、２つの主要な機能を提供するコードのブロックである。第１においては、通常はセキュリティ・ヘッダ１２６によって提供されるタイプの１ないしは複数のセキュリティ・チェックが、ブロック１２２の実行時に行われる。これらのチェックは、セキュリティ・ヘッダ１２６によって行われる二重チェックとするか、あるいはそれらへの追加とすることができる。セキュリティ・チェックをセキュリティ・ヘッダ１２６とブロック１２２の間に分ける方法は、広く多様な方法を用いて行うことが可能であり、それ自体は本発明の重要な側面ではない。

【００８６】

しかしながら、この例においては、アプリケーション１２４を暗号解除する機能がヘッダ１２６によって指示されることはなく、ブロック１２２のコントロールの下に置かれる。アプリケーション１２４の暗号解除は、好ましくはブロック１２２の実行時の最終動作とし、それに続いてブロック１２２から暗号解除後のアプリケーション１２４にコントロールが返される。

【００８７】

この方法によれば、ブロック１２２が、そのプロシージャの最後への到達に成功し、暗号解除が許可されて実行されるまで、アプリケーション１２４の暗号解除が生じることがあり得ない。

【００８８】

したがってセキュリティ・ブロック１２２は、本発明によって提供される保護を迂回しようとするハッカーのターゲットになりやすい。ブロック１２２に対する攻撃の成功は、次のようにして防止される。

【００８９】
セキュリティ・ブロック１２２は、当初（実行前）は、暗号化された形式で保持されている。暗号化は、暗号解除に少なくとも１つの変換鍵を要求する暗号化アルゴリズムによる。暗号解除に変換鍵を要求する暗号化アルゴリズムは、数多く知られている。変換鍵は、一般に数字となるか、あるいはアルゴリズム内において、変換鍵を構成する英数文字のマシン・コード値から数字形式に変換された後に数字形式で使用される。本発明とともに使用可能なアルゴリズムの１つのクラスは、ＣＲＣ（巡回冗長検査）アルゴリズムとして知られている。ＣＲＣアルゴリズムは、ターゲットのコード・ブロックを評価してターゲット・ブロックの特性である数値を生成する。つまり、単純なＣＲＣアルゴリズムは、ターゲットのコード・ブロックの各文字を表すマシン・コード値を互いに加算するものとすることができる。ほかのＣＲＣアルゴリズムは、単純な加算プロセスよりはるかに複雑になることもある。しかしながら、ここで重要なことは、ＣＲＣアルゴリズムがターゲットのコード・ブロックに適用されるごとに、同一の値が返されることであり、ターゲット・ブロック内に変更があると異なる値が返されることである。
【００９０】
この例においては、セキュリティ・ブロック１２２が、ＣＲＣ暗号化アルゴリズムの使用によって暗号化された形式に保持されている。したがって、最初にブロック１２２が実行されるときに暗号解除ステップが必要になる。この暗号解除は、ブロック１２２の残りを非実行可能形式から実行可能形式へ暗号解除することを要求する。これは、ターゲット・ブロックからＣＲＣ値として導出される変換鍵に基づく暗号解除ルーチンを実行することによって達成される。ターゲット・ブロックは、好ましくは暗号化されたブロック１２２とする。
【００９１】
これらの構成の結果は、アプリケーション１２４が呼び出されたときのイベントのシーケンスを説明することによってもっともよく理解することができる。
【００９２】
最初に、プロセッサ１２のプログラム・ポインタがセキュリティ・ヘッダ１２６の先頭にポイントされ、ブロック１２２にコントロールを渡す前にセキュリティ・チェックが実行される。この段階においては、まだアプリケーション１２４が暗号化された形式のままである。ブロック１２２もまた暗号化された形式であり、最初にその暗号解除が行われる。暗号解除は、変換鍵を必要とするアルゴリズムによって達成される。この変換鍵は、ブロック１２２自体から導出されたＣＲＣ値である。ブロック１２２が修正されていなければ、適切なＣＲＣ値が返されて変換が許される。その後、ブロック１２２を正常に実行することが可能になり、その結果としてブロック１２２の動作によってアプリケーション１２４が暗号解除されることになる。ブロック１２２が完全に実行された後は、コントロールがアプリケーション１２４に渡されて、通常の方法に従ってそれが実行される。しかしながら、ブロック１２２が何らかの形で修正されていた場合には、ブロック１２２のＣＲＣ値が変化している。その結果、暗号解除アルゴリズム内において変換鍵として使用される値が、正しい暗号解除に適切でなくなる。ブロック１２２の暗号解除されたバージョンは、改ざんされることになる。実際の状況においては、変換鍵に対するあらゆる変更が、それ以上にブロック１２２が実行されることを防止する充分な改ざんをもたらすと考えられる。これは、アプリケーション１２４を暗号化された形式のままに維持することになる。
【００９３】
デバッガ・プログラムを使用してブロック１２２を分析しようとするあらゆる試みは、前述したように、ブロック１２２内への割り込みコマンドの挿入をもたらす。したがって、デバッガを使用してブロック１２２を調べる行為は、デバッガが使用されて変更が行われる前であっても、それ自体がブロック１２２のＣＲＣ値を変更することになる。ＣＲＣ値に対する変更は、アプリケーション１２４の暗号解除を妨げる。このように、ブロック１２２によって提供されるセキュリティを調べる試みによって、セキュリティ・ブロックに対して、それを調べた結果としての修正がまったく行われない場合であっても、アプリケ

ーション１２４の動作が妨げられる。

【００９４】

セキュリティを迂回しようとするハッカーは、セキュリティ・チェックを受け持つと見られるコードのブロックの観察にもっとも関心を寄せることになる。したがって、セキュリティ・ブロック１２２を暗号解除するためのＣＲＣ値の生成に使用されるターゲット・ブロックが、アプリケーション１２４の暗号解除の認可に先行して１ないしは複数のセキュリティ・チェックを処理するブロックの少なくとも一部を含むことは好ましい。つまり、これらのセキュリティ・チェックのすべての実行が、暗号解除されたブロック１２２によって可能になり、それにおいてＣＲＣ変換鍵の導出が、暗号化されたブロック１２２の一部もしくは全部から可能になる。それに代えて、いくつかのチェックがブロック１２２によって実行されるようにしてもよく、その場合ＣＲＣ値を完全に、もしくは部分的に、ヘッダ１２６の全部もしくは一部から導出するようにもできる。

【００９５】

（例８）

図９は、例７のさらに複雑な変形を示している。図９に示されている多くの特徴は、図８の特徴に対応しており、それらの特徴には類似の番号が使用されている。

【００９６】

図８および９の具体化の間における主要な相違は、セキュリティ・ブロック１２２の暗号化の方法にある。ブロック１２２は、複数のサブ‐ブロック１３０として示されている。サブ‐ブロック１３０は、まとまって暗号化された形式のブロック１２２を提供するが、各サブ‐ブロック１３０は、独立に暗号化される。それぞれは、同一のアルゴリズムおよびそれぞれの変換鍵によって暗号化してもよく、またそれぞれが同一の変換鍵を要求する異なったアルゴリズムによって、もしくは異なる変換鍵を要求する異なったアルゴリズムによって暗号化してもよい。

【００９７】

図９の例の実行時は、図８に関連して説明した動作のシーケンスと概略で等しくなるが、次に述べる点が異なる。ブロック１２２が最初に呼び出されたとき、暗号解除アルゴリズムが実行されて最上位のサブ‐ブロック１３０だけが暗号解除される。これは、暗号解除が行われているサブ‐ブロック、もしくは別のサブ‐ブロック、あるいは別の場所にあるターゲット・ブロック（たとえばセキュリティ・ヘッダ１２６の全部もしくは一部）のＣＲＣ値を使用して行うことができる。暗号解除の後は、最初のサブ‐ブロック１３０の実行が可能になる。これは、セキュリティ・チェックもしくはそのほかの初期ルーチンの実施とすることができる。最初のサブ‐ブロック１３０の実行は、２番目のサブ‐ブロック１３０を暗号解除するインストラクションを伴って終了する。これは、変換鍵を、最初のサブ‐ブロック１３０の暗号解除に使用される変換鍵と同一とする場合であっても、好ましくは暗号解除の時点において導出される変換鍵を要求する暗号解除アルゴリズムによって達成される。つまり、最初のサブ‐ブロックの暗号解除以降におけるターゲット・ブロック内への介入がＣＲＣコードの変更を招き、その結果、２番目のサブ‐ブロック１３０の適正な暗号解除がもたらされない。

【００９８】

これらの動作をより完全な形で図１０に示す。図１０は、最初と２番目のサブ‐ブロック１３０Ａ、１３０Ｂの間の境界における３行のコードを示している。サブ‐ブロック１３０Ａは、第９９行において最終行を伴って終了するが、その性質は本発明にとって重要ではない。続いて第１００行が実行される。これは、暗号解除インストラクションである。このインストラクションは、暗号解除予定のデータのブロック、すなわちここでは第１０１～２００行を指定し、それは２番目のサブ‐ブロック１３０Ｂを表している。インストラクション１００は、使用されるアルゴリズムも識別し、ここではそれがアルゴリズムＡである。最後にインストラクション１００は、アルゴリズムＡ内において使用する変換鍵を導出するためのターゲット・ブロックとして使用される行を指定（ここでは第１０１～２００行を指定）することによって、使用されるべき変換鍵を明示する。

【００９９】
第１００行が実行されると、実行が第１０１行に移る。第１００行が暗号解除の成功に到達していれば、第１０１行が、２番目のサブ‐ブロック１３０Ｂの最初の動作を適正に示すことになる。この暗号解除の成功は、サブ‐ブロック１３０Ｂが修正されてなく、その結果、正しいＣＲＣ値が暗号解除鍵として使用された場合に得られる。しかしながら、２番目のサブ‐ブロック１３０Ｂが修正されていた場合、あるいはＣＲＣ値の計算が行われる時点においてデバッガ・プログラムによる検査が行われている場合には、正しくないＣＲＣ値が第１０１行の暗号解除に使用され、サブ‐ブロック１３０Ｂが改ざんされて、そのポイントにおいて実行が停止する。その場合、アプリケーション１２４は、暗号化された形式のままとなる。

【０１００】
セキュリティ・ブロック１２２の実行が継続するに従って、前述の方法を用いて図９の矢印１３２に示されるようにサブ‐ブロックを進めることによって、各サブ‐ブロック１３０が順番に暗号解除される。ブロック１２２内に改ざんがなく、かつデバッガ・プログラムが使用されていない場合には、ブロック１２２が完全に改ざんのない形に暗号解除され、実行が図１１に示されている最終サブ‐ブロック１３０の末尾への到達に成功する。最終サブ‐ブロック１３０の末尾、すなわちセキュリティ・ブロック１２２の実行の最後には、アプリケーション１２４を暗号解除するインストラクション（第９９８行）およびそれに続くアプリケーション１２４へ移動するインストラクション（第９９９行）がある。９９８における暗号解除は、本発明の原理をさらに適用するために、前述したようなＣＲＣ値として導出された暗号解除鍵を必要とするアルゴリズムを基礎とすることができる。第９９９行へは、各サブ‐ブロック１３０が適正に暗号解除された場合に限って到達し、適正に実行される。

【０１０１】
（例９）
前述した例７および８は、図８および９に示されるように、ソフトウエアの実行がＲＡＭ２０に個別にインストールされるセキュリティ・ブロック１２２を伴って開始することを前提としている。図１２Ａおよび１２Ｂに示されている第３の例は、次に説明するように、さらに暗号解除ならびに伸張ステップを含む。

【０１０２】
最初にＲＡＭ２０は、単一ブロックのソフトウエア１２０を含んでいる（図１２Ａ）。これは、暗号化されたアプリケーション１２４、セキュリティ・ヘッダ１２６、およびこの例においてはアプリケーション１２４の末尾に取り付けられる形で示されているセキュリティ・スタブ(stub)１４０を組み込んでいる。セキュリティ・スタブ１４０は、例７または８のセキュリティ・ブロック１２２の圧縮ならびに暗号化の行われたバージョンを組み込んでいる。セキュリティ・ヘッダ１２６は、スタブ１４０を暗号解除および／または伸張し、結果を、セキュリティ・ブロック１２２を形成する独立したブロック（図１２Ｂ）としてインストールするインストラクションを含む。セキュリティ・ヘッダ１２６が完全に実行されると、コントロールがセキュリティ・ブロック１２２に渡される。続く実行は、例７および８との関連から前述した説明と同様に進行することになる。

【０１０３】
この例の利点は、正当なユーザに対するソフトウエア・ライセンスの一部としてスタブ１４０が提供されることを可能にすることであり、その結果、ライセンスされたアプリケーションがインストールされると、ブロック１２０が、図９Ａに示されるようにスタブ１４０を含む形でインストールされる。これは、アプリケーション１２４の適正にライセンスされたバージョンに、常にセキュリティ・スタブ１４０が伴うことを保証し、したがって本発明のソフトウエア保護構成の恩典によって、認可されたコピーが保護された状態に保たれることを保証する。

【０１０４】
詳細に述べれば、これらの例においては、暗号解除およびＣＲＣ値の計算のための多くの

各種アルゴリズムを使用することができる。

【０１０５】

（要約）

これまで説明した例のそれぞれにおいて、実行可能コードの実行間にＲＡＭ内のコードが変化する。最も簡単な例（例１）においては、コードが１回だけ変化してルーチンを作成する。別の例においては、コードがより頻繁に変化し、いくつかの例においては、悪意のあるウォッチャが、絶えず変化するように現れるコードを見ることになる。

【０１０６】

いずれの場合においても、被保護データが、当初は使用可能な形式で存在しない。しかしながら、ほかの実行可能コードが被保護コードに関連付けされており、被保護データに関する情報を組み込んでいる。この情報は、そのロケーション、暗号化テクニックもしくは鍵、暗号化鍵のロケーション、あるいは暗号化された形式のコード全体に関係する情報とすることができる。これは、別の実行可能コードが別のコードを生成すること、すなわち保護された状態から被保護コードを解放し、使用可能な形式でのアクセスを許可するコードを生成することを可能にする。

【０１０７】

上記の構成には、非常に多くの変形および修正を行うことが可能である。詳細に述べれば、前述の説明の完全な理解から、そこに説明されている各種のテクニックを各種の組み合わせにおいて使用可能なことが容易に明らかになろう。たとえば、例３に関して説明した配置変更の原理を例５に組み込み、それにより一時的にのみ生成される例５のステップに加えて、それらが生成されるロケーションを、好ましくはランダムもしくは擬似ランダムに変更することが可能である。

【０１０８】

これまで示した例においては、被保護データが実行可能であったが、このテクニックは非実行可能データ・ファイルに対しても容易に適応することができる。

【０１０９】

以上の明細においては、特に重要であると思われる本発明の特徴に注意が注がれるべく努めているが、出願人が、特定の強調が置かれているか否かによらず、図面を参照した、および／またはそれに示されている上記のあらゆる特許性のある特徴、もしくは特徴の組み合わせに関する保護を主張していることが理解されるものとする。

【図面の簡単な説明】

【図１】　本発明の実施における使用のための高度に単純化した形式のコンピュータ・システムを示している。

【図２】　ＲＡＭ内にソフトウエアをストアするための周知の構成の高度に略図化した表現である。

【図３】　本発明に従ったＲＡＭの使用を示している。

【図４】　本発明に従った、配置変更構成を組み込んだＲＡＭの使用を示している。

【図５】　本発明に従った、埋め込みセキュリティ・プロシージャを組み込んだＲＡＭの使用を示している。

【図６】　本発明に従った、自己修正構成を組み込んだＲＡＭの使用を示している。

【図７】　本発明に従った、被保護データの部分的な暗号解除に関する構成を組み込んだＲＡＭの使用を示している。

【図８】　分離セキュリティ・ブロックを使用する本発明の別の例を実施するシステム内におけるＲＡＭの内容を示した概略図である。

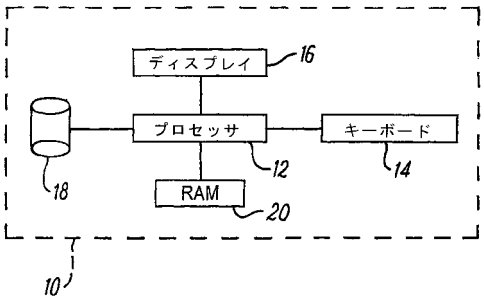
【図９】　図８に対応しており、図８の例の変形を示している。

【図１０】　図８および９のセキュリティ・ソフトウエアのリスティングの一部を略図的に示している。

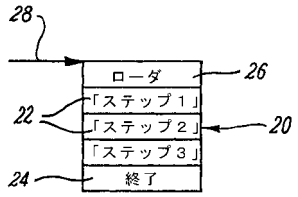
【図１１】　図８および９のセキュリティ・ソフトウエアのリスティングの末尾を略図的に示している。

【図１２】　本発明の実施の別の形式を略図的に示している。

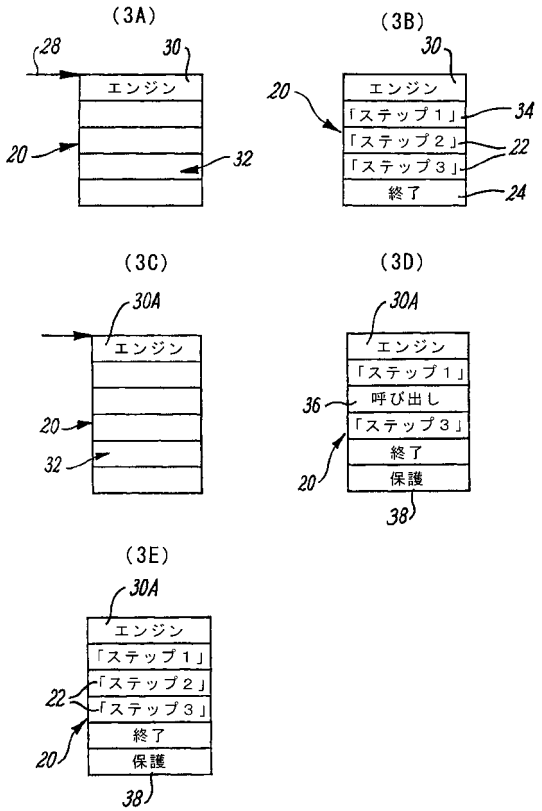
【図１】


ディスプレイ
16
プロセッサ
キーボード
12
14
18
RAM
20
10

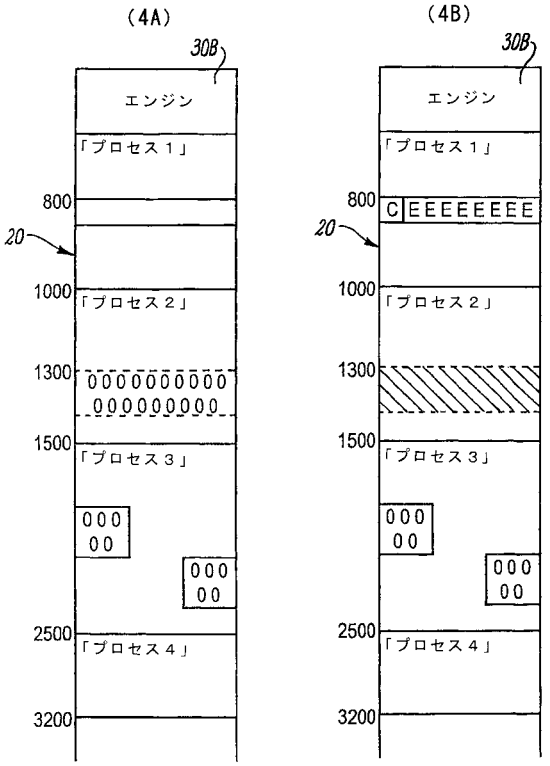
【図２】


28
ローダ
26
「ステップ１」
22
「ステップ２」
20
「ステップ３」
24
終了

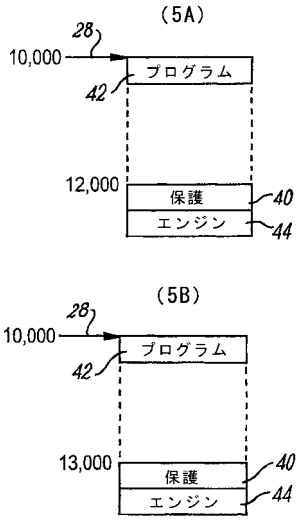
【図３】


(3A)
28
30
エンジン
20
32
(3B)
30
エンジン
20
「ステップ１」
34
「ステップ２」
22
「ステップ３」
終了
24
(3C)
30A
エンジン
20
32
(3D)
30A
エンジン
「ステップ１」
36
呼び出し
「ステップ３」
20
終了
保護
38
(3E)
30A
エンジン
「ステップ１」
22
「ステップ２」
「ステップ３」
20
終了
保護
38

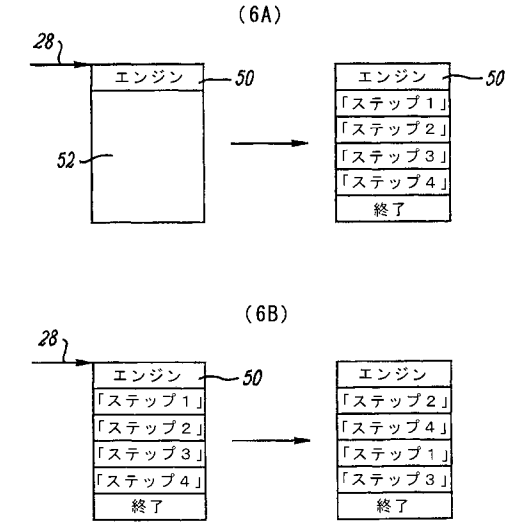
【図４】


(4A)
30B
エンジン
「プロセス１」
800
20
1000
「プロセス２」
1300
1500
「プロセス３」
2500
「プロセス４」
3200
(4B)
30B
エンジン
「プロセス１」
800
C EEEEEEEE
20
1000
「プロセス２」
1300
1500
「プロセス３」
2500
「プロセス４」
3200

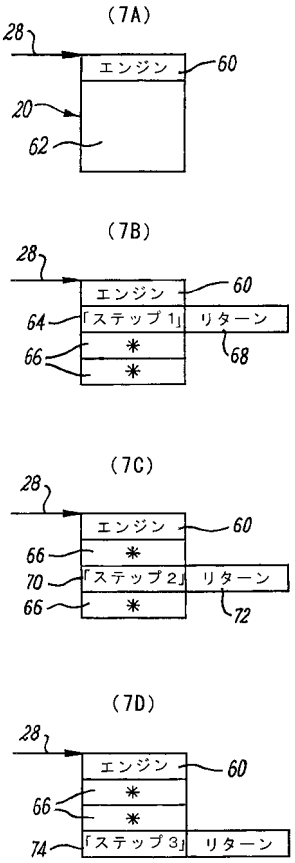
【図５】


(5A)
28
10,000
プログラム
42
12,000
保護
40
エンジン
44
(5B)
28
10,000
プログラム
42
13,000
保護
40
エンジン
44

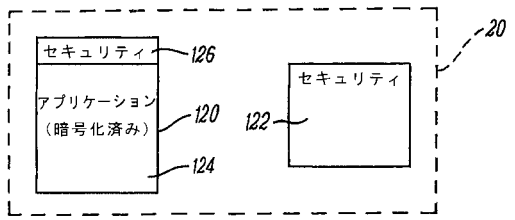
【図６】



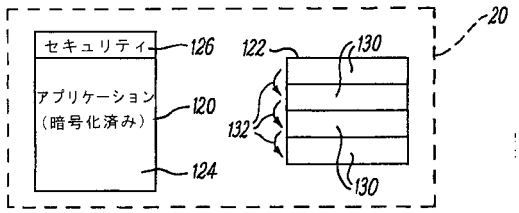
【図７】



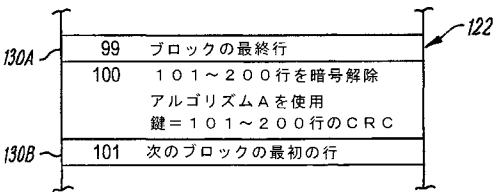
【図８】



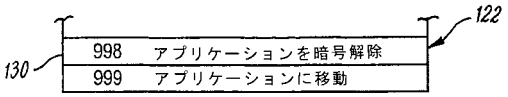
【図９】



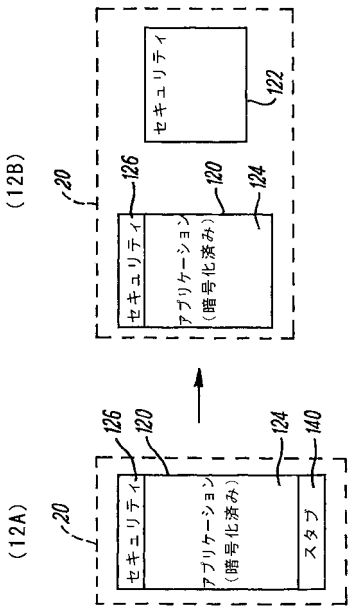
【図１０】



【図１１】



【図１２】



フロントページの続き

審査官　深沢　正志

(56)参考文献　米国特許第０６００６３２８（ＵＳ，Ａ）
特開平０４－１６３６２７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 21/00 - 21/24