(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5689472号
(P5689472)
(45)発行日　平成27年3月25日(2015.3.25)　(24)登録日　平成27年2月6日(2015.2.6)

(51)Int.Cl.　　F I
G06F 21/12 (2013.01)　　G06F 21/12 310
G06F 21/14 (2013.01)　　G06F 21/14

請求項の数 49　(全 32 頁)

(21)出願番号　特願2012-538154(P2012-538154)
(86)(22)出願日　平成22年11月12日(2010.11.12)
(65)公表番号　特表2013-511077(P2013-511077A)
(43)公表日　平成25年3月28日(2013.3.28)
(86)国際出願番号　PCT/CA2010/001761
(87)国際公開番号　WO2011/057393
(87)国際公開日　平成23年5月19日(2011.5.19)
審査請求日　平成25年10月25日(2013.10.25)
(31)優先権主張番号　61/260,887
(32)優先日　平成21年11月13日(2009.11.13)
(33)優先権主張国　米国(US)

(73)特許権者　510302168
イルデト　カナダ　コーポレーション
カナダ国、オンタリオ州　ケー２ケー　３ジー５、オタワ、スイート　３００、ソーラント　ロード　２５００
２５００　Ｓｏｌａｎｄｔ　Ｒｏａｄ，Ｓｕｉｔｅ　３００，　Ｏｔｔａｗａ，Ｏｎｔａｒｉｏ　Ｋ２Ｋ　３Ｇ５　ＣＡＮＡＤＡ
(74)代理人　100104444
弁理士　上羽　秀敏
(74)代理人　100112715
弁理士　松山　隆夫
(74)代理人　100125704
弁理士　坂根　剛

最終頁に続く

(54)【発明の名称】悪意ある実行環境内での静的および動的攻撃からＪａｖａバイトコードを保護するシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
Ｊａｖａバイトコードのタンパリング耐性を向上させるためのシステムであって、
１つまたは複数のプロセッサと、
前記１つまたは複数のプロセッサの少なくとも１つに操作可能に結合され、前記１つまたは複数のプロセッサの少なくとも１つに実行された際に、前記１つまたは複数のプロセッサの少なくとも１つに、以下の処理を実行させる命令を格納した、１つまたは複数のメモリと、
前記処理は、
安全なＪａｖａバイトコードと対応するセキュリティ情報を生成するために、構築時、Ｊａｖａバイトコードに対して保護を適用することと、
前記安全なＪａｖａバイトコードの少なくとも一部をＪａｖａヴァーチャルマシン（ＪＶＭ）にロードすることと、
前記安全なＪａｖａバイトコードのロード時と実行時にＪａｖａネイティブインタフェース（ＪＮＩ）ブリッジを介して前記ＪＶＭと通信するために、デプロイ時に実行するよう構成されたソフトウエアモジュールを含むセキュリティモジュールに、前記対応するセキュリティ情報をロードすることと、
１つまたは複数の保護機構を介して、前記Ｊａｖａバイトコードのローディングと実行の間、前記Ｊａｖａバイトコードへの静的および動的攻撃に対抗すること、とを含み、
前記１つまたは複数の保護機構の少なくとも１つは、前記セキュリティモジュールに組

み込まれ、前記静的および動的攻撃は、前記セキュリティモジュールにロードされた前記対応するセキュリティ情報の少なくとも一部に基づいて対抗され、前記セキュリティモジュールは、前記ＪＶＭと共に前記ＪＮＩブリッジを介して前記安全なＪａｖａバイトコードと共に実行されるよう構成されている、システム。

【請求項２】

前記Ｊａｖａバイトコードが、構築時、複数の部分に分けられ、実行時、前記複数の部分が前記ＪＶＭの異なる部分におよび前記セキュリティモジュールに対して実行される、請求項１に記載のシステム。

【請求項３】

前記安全なＪａｖａバイトコードが、被保護Ｊａｖａアプリケーションバイトコードスタブ、被保護アプリケーションペイロード、および暗号化されたクラスバイトコードフレームを含む、請求項１に記載のシステム。

【請求項４】

前記被保護Ｊａｖａアプリケーションバイトコードスタブを介して、前記被保護アプリケーションペイロードが起動される、請求項３に記載のシステム。

【請求項５】

前記セキュリティモジュールが、被保護バイトコードクラスローダを含み、

前記被保護バイトコードクラスローダを使用して前記暗号化クラスバイトコードフレームにより前記被保護アプリケーションペイロードが起動される、請求項３または４に記載のシステム。

【請求項６】

前記セキュリティモジュールが、ＪＶＭ環境への機能的延長部であり、被保護Ｊａｖａアプリケーションの信頼の基礎を提供するよう構成され、かつ、複数の異なる安全なＪａｖａバイトコードに整合するよう構成されている、請求項１に記載のシステム。

【請求項７】

前記セキュリティモジュールがローカルハードウェアごとの保護技術およびシステムを適用するネィティブプログラミング言語で書かれる、請求項１から６のいずれかに記載のシステム。

【請求項８】

前記セキュリティモジュールが、ローカルネィティブソフトウェアごとの保護技術およびシステムを適用するネィティブプログラミング言語で書かれる請求項１から６のいずれかに記載のシステム。

【請求項９】

前記セキュリティモジュールが、ホワイトボックスセキュリティ技術を用いて保護される、請求項１から６のいずれかに記載のシステム。

【請求項１０】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構のパラメータが、ユーザの好みにより構成可能で、かつ、前記ユーザの好みに従って前記安全なＪａｖａバイトコードが生成される、請求項１から９のいずれかに記載のシステム。

【請求項１１】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が前記Ｊａｖａバイトコードの実行の一部を前記セキュリティモジュールへ移動させ、それにより前記ＪＶＭが実行時に前記Ｊａｖａバイトコードのすべてを実行するわけではなくなり、元のＪａｖａバイトコードが、ＪＶＭ上で完全に観察可能にはならない、請求項１０に記載のシステム。

【請求項１２】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、前記Ｊａｖａバイトコードの選択されたメソッドを、ＪＶＭには直接可視的でなく、セキュリティモジュールによってのみ起動可能であるネィティブ形式の機能に変換する、請求項１０に記載のシステム。

【請求項１３】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が前記Ｊａｖａバイトコードに対してデータフロー変換を行い、機能性を変更することなしに、前記Ｊａｖａバイトコードのコード構造を変換する、請求項１０に記載のシステム。

【請求項１４】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、前記Ｊａｖａバイトコードに対する制御フロー変換を行い、機能性を変更することなく前記Ｊａｖａバイトコードのコード構造を変換する、請求項１０に記載のシステム。

【請求項１５】

前記セキュリティモジュールが、実行されるべきＪａｖａバイトコードを、ＪＶＭの作業空間内にジャストインタイムでロードしかつ復元し、かつその後実行後の復元したＪａｖａバイトコードを除去する、請求項１０に記載のシステム。

【請求項１６】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、アンチデバッギングモニタリングを行う、請求項１から１０のいずれかに記載のシステム。

【請求項１７】

前記少なくとも１つの保護機構が、

カーネルからのそれ自らのプロセスマップを周期的にチェックし、

防御アクションをトリガすることによりそのメモリ空間内へロードされるＪａｖａのデバッギングに関連するライブラリに応答する、請求項１６に記載のシステム。

【請求項１８】

前記応答が、

防御アクションをトリガすることにより、そのメモリ空間内へロードされるＪＤＰＡ（Ｊａｖａ　Ｐｌａｔｆｏｒｍ　Ｄｅｂｕｇ　Ａｒｃｈｉｔｅｃｔｕｒｅ）に関連するライブラリに応答するステップを含む、請求項１７に記載のシステム。

【請求項１９】

前記応答が、

防御アクションをトリガすることにより、そのメモリ空間内へロードされるＪＶＭＴＩ（Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ　Ｔｏｏｌ　Ｉｎｔｅｒｆａｃｅ）に関連するライブラリに応答するステップを含む、請求項１７に記載のシステム。

【請求項２０】

前記少なくとも１つの保護機構が、デバッギングスレッドの起動を監視し、かつ

防御アクションをトリガすることにより起動されるデバッギングスレッドに応答する、請求項１６に記載のシステム。

【請求項２１】

前記少なくとも１つの保護機構が、デバッギングツールにより実行されるラインブレークメッセージを監視し、かつ

防御アクションをトリガすることによりラインブレークメッセージの検出に応答する、請求項１６に記載のシステム。

【請求項２２】

前記１つまたは複数の保護機構は、ホワイトボックス（ＷＢ）静的セキュリティハンドラを含み、

前記ＷＢ静的セキュリティハンドラは、

ユーザからの暗号キーを含む暗号情報を受けて、

構築時に使用されるＷＢ暗号化キーデータと、

実行時、前記セキュリティモジュールにより使用されるＷＢ復号化キーデータおよびＷＢセキュリティモジュールユーティリティとを生成するものである、請求項１から１０のいずれかに記載のシステム。

【請求項２３】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、バイトコード完全性検証（ＢＩＶ）システムを含み、

構築時に前記安全なＪａｖａバイトコードのハッシュ値が計算され、かつ、
前記タンパリング耐性セキュリティモジュールが、前記構築時ハッシュ値が実行時に計算される前記ハッシュ値に等しいかどうか決定し、前記セキュリティモジュールが、検証失敗に応答してタンパリング対抗策を起動する、請求項１から１０のいずれかに記載のシステム。
【請求項２４】
前記セキュリティモジュールが、
前記ＪＶＭで、インタフェースを介してＪａｖａバイトコードを入手し、
前記バイトコードの動的ハッシュ値を計算し、
前記構築時ハッシュ値が実行時に計算した前記ハッシュ値に等しいかどうか決定し、等しくない場合にはタンパリング抵抗策を起動するよう動作することができる、請求項２３に記載のシステム。
【請求項２５】
前記ハッシュ値検証が、タンパリング耐性ゲートキーパーにより行われる、請求項２４に記載のシステム。
【請求項２６】
前記ハッシュ値検証が、構築時と実行時のハッシュ値を明示的に比較することなしに行われる、請求項２４に記載のシステム。
【請求項２７】
前記ハッシュ値検証が、前記Ｊａｖａバイトコードの選択されたクラスおよびメソッドに対してのみ行われる、請求項２５に記載のシステム。
【請求項２８】
前記ハッシュ値検証ならびに選択されたクラスおよびメソッドに関連するデータが暗号化される、請求項２７に記載のシステム。
【請求項２９】
静的ハッシュ値が、異なるキーデータを使用したＷＢサイファーにより暗号化される、請求項２７に記載のシステム。
【請求項３０】
ＢＩＶシステムが閉じる際に、前記セキュリティモジュールが、ＢＩＶシステムにより使用された関連のメモリ空間および他の情報をクリーンにする、請求項２７に記載のシステム。
【請求項３１】
前記１つまたは複数の保護機構が、構築時、被保護Ｊａｖａアプリケーションバイトコードスタブ、被保護アプリケーションペイロード、および暗号化クラスバイトコードフレームを生成するためのセキュアローディングバイトコード（ＳＬＢ）ツールを含み、
前記セキュリティモジュールが、ＳＬＢ動的セキュリティハンドラを含み、このＳＬＢ動的セキュリティハンドラが、
前記安全なＪａｖａアプリケーションバイトコードに対応する前記暗号クラスバイトコードフレームをメモリバッファへロードし、
前記暗号化クラスに対応するホワイトボックス被保護復号化キーデータを介して前記暗号化クラスバイトコードフレーム内に含まれる各暗号化クラスを復号化し、かつ
各復号化したクラスバイトコードを、セキュリティモジュールクラスローダーによりアプリケーション作業空間へロードして、前記アプリケーション作業空間内の前記Ｊａｖａアプリケーションバイトコードを実行するためのものである、請求項１から１０のいずれかに記載のシステム。
【請求項３２】
前記安全なＪａｖａバイトコードの実行時のどの時点においても、当該安全なＪａｖａバイトコードの一部のみが、復号された形式で格納される、
請求項１から１０のいずれかに記載のシステム。
【請求項３３】

Ｊａｖａバイトコードのタンパリング耐性を向上させるために、１つまたは複数のコンピュータデバイスで実行される方法であって、
前記１つまたは複数のコンピュータデバイスの少なくとも１つにより、構築時に、Ｊａｖａバイトコードに保護を適用して安全なＪａｖａバイトコードと対応するセキュリティ情報とを生成するステップと、
前記１つまたは複数のコンピュータデバイスの少なくとも１つにより、前記安全なＪａｖａバイトコードの少なくとも一部をＪａｖａヴァーチャルマシン（ＪＶＭ）にロードするステップと、
前記安全なＪａｖａバイトコードのロード時と実行時にＪａｖａネイティブインタフェース（ＪＮＩ）ブリッジを介して前記ＪＶＭと通信するために、前記１つまたは複数のコンピュータデバイスの少なくとも１つにより、デプロイ時に実行するよう構成されたソフトウエアモジュールを含むセキュリティモジュールに、前記対応するセキュリティ情報をロードするステップと、
前記１つまたは複数のコンピュータデバイスの少なくとも１つにより、１つまたは複数の保護機構を介して、前記Ｊａｖａバイトコードのローディングと実行の間、前記Ｊａｖａバイトコードへの静的および動的攻撃に対抗するステップ、とを含み、
前記１つまたは複数の保護機構の少なくとも１つは、前記セキュリティモジュールに組み込まれ、前記静的および動的攻撃は、前記セキュリティモジュールにロードされた前記対応するセキュリティ情報の少なくとも一部に基づいて対抗され、前記セキュリティモジュールは、前記ＪＶＭと共に前記ＪＮＩブリッジを介して前記安全なＪａｖａバイトコードと共に実行されるよう構成されている、方法。
【請求項３４】
前記Ｊａｖａバイトコードが、構築時、複数の部分に分けられ、実行時、前記複数の部分が前記ＪＶＭの異なる部分におよび前記セキュリティモジュールに対して実行される、請求項３３に記載の方法。
【請求項３５】
前記安全なＪａｖａバイトコードが、被保護Ｊａｖａアプリケーションバイトコードスタブ、被保護アプリケーションペイロード、および暗号化されたクラスバイトコードフレームを含む、請求項３３に記載の方法。
【請求項３６】
前記被保護Ｊａｖａアプリケーションバイトコードスタブを介して、前記被保護アプリケーションペイロードが起動される、請求項３５に記載の方法。
【請求項３７】
前記セキュリティモジュールが、被保護バイトコードクラスローダを含み、
前記被保護バイトコードクラスローダを使用して前記暗号化クラスバイトコードフレームにより前記被保護アプリケーションペイロードが起動される、請求項３５または３６に記載の方法。
【請求項３８】
前記セキュリティモジュールが、ＪＶＭ環境への機能的延長部であり、被保護Ｊａｖａアプリケーションの信頼の基礎を提供するよう構成され、かつ、複数の異なる安全なＪａｖａバイトコードに整合するよう構成されている、請求項３３に記載の方法。
【請求項３９】
前記セキュリティモジュールがローカルハードウェアごとの保護技術およびシステムを適用するネィティブプログラミング言語で書かれる、請求項３３から３８のいずれかに記載の方法。
【請求項４０】
前記セキュリティモジュールが、ローカルネィティブソフトウェアごとの保護技術およびシステムを適用するネィティブプログラミング言語で書かれる、請求項３３から３８のいずれかに記載の方法。
【請求項４１】

前記セキュリティモジュールが、ホワイトボックスセキュリティ技術を用いて保護される、請求項３３から３８のいずれかに記載の方法。

【請求項４２】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構のパラメータが、ユーザの好みにより構成可能で、かつ、前記ユーザの好みに従って前記安全なＪａｖａバイトコードが生成される、請求項３３から４１のいずれかに記載の方法。

【請求項４３】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が前記Ｊａｖａバイトコードの実行の一部を前記セキュリティモジュールへ移動させ、それにより前記ＪＶＭが実行時に前記Ｊａｖａバイトコードのすべてを実行するわけではなくなり、元のＪａｖａバイトコードが、ＪＶＭ上で完全に観察可能にはならない、請求項４２に記載の方法。

【請求項４４】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、前記Ｊａｖａバイトコードの選択されたメソッドを、ＪＶＭには直接可視的でなく、セキュリティモジュールによってのみ起動可能であるネィティブ形式の機能に変換する、請求項４２に記載の方法。

【請求項４５】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が前記Ｊａｖａバイトコードに対してデータフロー変換を行い、機能性を変更することなしに、前記Ｊａｖａバイトコードのコード構造を変換する、請求項４２に記載の方法。

【請求項４６】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、前記Ｊａｖａバイトコードに対する制御フロー変換を行い、機能性を変更することなく前記Ｊａｖａバイトコードのコード構造を変換する、請求項４２に記載の方法。

【請求項４７】

前記セキュリティモジュールが、実行されるべきＪａｖａバイトコードを、ＪＶＭの作業空間内にジャストインタイムでロードしかつ復元し、かつその後実行後の復元したＪａｖａバイトコードを除去する、請求項４２に記載の方法。

【請求項４８】

前記１つまたは複数の保護機構中の少なくとも１つの保護機構が、アンチデバッギングモニタリングを行う、請求項３３から４２のいずれかに記載の方法。

【請求項４９】

少なくとも１つのコンピュータ読取り可能な記録媒体であって、１つまたは複数のコンピュータデバイスで実行されたときに、前記１つまたは複数のコンピュータデバイスの少なくとも１つに、請求項３３から４８のいずれかに記載の方法を実行させるコンピュータ読取り可能な命令を記録した、記録媒体。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、一般にコンピュータソフトウェアに関し、かつより詳細には、コンピュータソフトウェアに、悪意ある実行環境内での静的および動的攻撃に対する耐性をつける方法及びシステムに関する。

【背景技術】

【０００２】

コンピュータプログラミング産業においては、Ｊａｖａプログラミング言語は、あらゆる主要な産業分野で使用され、広範なデバイス、コンピュータおよびネットワーク内に存在する。Ｊａｖａアプリケーションは、Ｊａｖａプログラミング言語で書かれ、Ｊａｖａヴァーチャルマシン（ＪＶＭ）上で実行される機種既存性のないバイトコードにコンパイルされる。ＪＶＭは、ホストのオペレーティグシステム（ＯＳ、Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ）およびホストのコンピュータプロセッシングユニット（ＣＰＵ、Ｃｏｍｐｕ

ｔｅｒ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）インストラクションセットアーキテクチャ（ＩＳＡ、Ｉｎｓｔｒｕｃｔｉｏｎ　Ｓｅｔ　Ａｒｃｈｔｅｃｔｕｒｅ）上に展開される。Ｊａｖａ技術は、その汎用性、効率、プラットフォームポータビリティおよびセキュリティにより、ネットワークコンピューティングのための理想的技術となっている。Ｊａｖａプログラミング言語は、ラップトップからデータセンター、ゲームコンソールから科学系スーパーコンピュータ、また携帯電話からインターネットまであらゆる分野に使用される。実際、可搬性、拡張性、汎用性および信頼性がＪａｖａの主要な強みである。しかしながら、このような遍在性はまたハッカーや関連のコンピュータ攻撃に対して十分な機会を与えてしまう。

【０００３】

信頼できないアプリケーションによるＪａｖａ環境への攻撃や不正アクセスを防止するために、Ｊａｖａ技術は、ウィルスやマルウェアなどの汚染されたソフトウェアが違法にダウンロードされたりインストールされたりする可能性があるホストマシンまたはデバイスの実行環境を保護するための、サンドボックスセキュリティモデルを含む。このような悪意ある攻撃を防止することは、電気通信システム、運輸システム、防衛システム、産業オートメーションシステムおよび電力管理システム等の高度に保護された環境およびシステム上で通常実行される重要なアプリケーションを設計する上で必須である。毎年、Ｊａｖａプログラミング言語を用いてこれら重要なシステムが、次々に設計実現される。

【０００４】

同様に、消費者向け電子機器産業も新しい時代を迎え、先進技術および製品、急激なメディアのデジタル化に対する要求および下落が続く消費者向け電子機器の価格に、新興市場での可処分所得の増大があいまって、前例のないスピードと範囲で消費者向け電子機器市場の成長に拍車がかかっている。これら消費者向け電子機器製品の多くは、ソフトウェアアプリケーションに依存して機能する。いくつかのＪａｖａプログラミング言語の強み（可搬性、拡張性、汎用性、信頼性および簡便性等）によって、消費者向け電子機器製品の全体的な開発およびデプロイにかかるコストが下がるため、より多くのＪａｖａ系プラットフォームおよびアプリケーションが新たな消費者向け製品に必ずデプロイされるようになっている。

【０００５】

ほとんどすべての消費者向け電子機器が、信用できない環境でも機能することを必要とする。信用できない環境で、消費者向け電子機器内のソフトウェアは、有益な理由（必要なサービスを受ける等）から望ましくない理由（機器をハッキングするため等）まで様々な目的で直接アクセスされ得る。その結果、以前にもまして多くのコンピュータアプリケーションが、相対的に悪意ある環境で実行される。たとえば、携帯装置（携帯メディアプレイヤーまたはスマートホン等）、ホームネットワーキング（セットトップボックス、メディアプレイヤーまたはパソコン等）およびウェブ系環境という領域は、攻撃者が長時間多くのリソースを費やすことが多い領域である。したがって、攻撃してくるソフトウェアから正当なソフトウェアを守ることは、激化する軍備拡張競争のようになりつつある。その上、高性能ハードウェアと複雑な攻撃ツールにより、侵入者は多くの新しい有利点を備える。

【０００６】

ソフトウェアのディストリビュータは、ソフトウェアが攻撃に対して堅牢で耐性が必ずあるようにする必要がある。しかしながら、所与のプラットフォームおよびソフトウェアは、時間、リソース、ツールおよび自由に使えるウェブ上の専門家をすべて備えている攻撃者には周知であることが多い。このような悪意ある攻撃の環境をよく「ホワイトボックス」環境と呼び、このような環境では、内容のすべてが丸見えで、したがって直接のアクセスおよびタンパリングの対象となる。これは、内容が隠されるか、または攻撃から保護されている、信頼があり保護された環境である「ブラックボックス」環境とは逆である。主流となっているホワイトボックス環境の悪意ある環境において、ソフトウェア系への直接かつ自動的な攻撃を防止または止めることが、セキュリティ上の最も厳しい課題の１つ

になりつつある。さらに、ホワイトボックス攻撃のための強固な防御を行い、適正かつセキュアな装置機能を確保する必要がある。Ｊａｖａプログラミング言語は、このようなセキュリティ上の問題および課題に取り組むために十分な設計になっていない。これに関しては、実際のところ、いくつかのＪａｖａの強みが、ＣまたはＣ＋＋によるプログラミングに比較して、セキュリティ上の脆弱性を引き起こしている。

【０００７】

Ｃ／Ｃ＋＋コードを生のバイナリデータに働きかつ目的のハードウェア（×８６またはパワーＰＣ等）に特定の低レベルの命令セットにコンパイルするＣ／Ｃ＋＋コンパイラとは違い、Ｊａｖａコンパイラは、Ｊａｖａソースコードを、ＪＶＭが実行時に翻訳できるクラスおよびプリミティブなタイプに働くより高いレベルのポータブルバイトコードにコンパイルする。プラットフォーム依存性は、ＪＶＭ内にカプセル化し、Ｊａｖａアプリケーションからは切り離される。

【０００８】

その上、標準的なＪａｖａコンパイラは、Ｃ／Ｃ＋＋コンパイラに共通してかつ通常みられるコンパイル時最適化を実行しない。その代り、Ｊａｖａは、より良い性能のために実行プロファイルを考慮に入れながら実行時にすべての最適化を行うジャストインタイム（ＪＩＴ、Ｊｕｓｔ－Ｉｎ－Ｔｉｍｅ）コンパイルに依存する。主なＣ／Ｃ＋＋コード最適化は、コンパイル時に実行される。たとえばインラインサブスティチューションによりバイナリ画像の周辺に散乱する所与の（メンバー）機能のコピーが得られる。すなわち、プリプロセッサを表現のコンパイル時評価と組み合わせて使用すれば、ソースコードに規定される定数の痕跡を残さないかも知れない等である。一般に、複雑に最適化されたコードをリバースエンジニアするのはより困難である。

【０００９】

さらに、Ｊａｖａプログラム依存性は、クラスがロードされる実行時に解決される。したがって、クラスの名称ならびにそのメソッドおよびフィールドの名称は、すべてのインポートされたクラス、呼び出されたメソッドおよびアクセスされたフィールドの名称とともに、クラスファイルの中に存在するはずである。一方で、Ｃ／Ｃ＋＋プログラムは静的にリンクされる。したがって、動的ライブラリからエクスポートされた名称を除いては、クラスの名称、メンバーおよび変数は、コンパイルされかつリンクされるプログラム内に存在する必要がない。

【００１０】

最後に、Ｊａｖａアプリケーションは、Ｊａｖａアーカイブ（ＪＡＲ、Ｊａｖａ　Ａｒｃｈｉｖｅ）ファイルのセットとして送付される。ＪＡＲフォーマットにより、複数のファイルを、本質的に非暗号化アーカイブで個別のクラスを比較的容易に取り出せる単独のアーカイブファイルにバンドルすることができる。これに比べて、Ｃ／Ｃ＋＋アプリケーションは、いくつかの動的ライブラリとリンクし得るモノリシックな実行可能ファイルとして配布されるので、プログラム情報および個別のコードを識別するのはそれほど容易ではない。

【００１１】

したがって、ＪａｖａバイトコードをＪａｖａソースへデコンパイルすることは、Ｃ／Ｃ＋＋をディスアセンブルするより簡単かつ容易で、したがって完全に自動化することも可能である。クラスヒエラルキー、ステートメント、クラスの名称、メソッドおよびフィールド等のプログラム情報をバイトコードからすべて回収することができる。多くのフリーウェアおよび商用のＪａｖａ難読化ツールが入手可能だが、いずれもバイトコードの実行に対する直接攻撃を阻止する保護を提供しない。結果として、現在のところ、Ｊａｖａリバースエンジニアリングは、日常茶飯事となっている。

【００１２】

さらに、ＪＶＭは、Ｊａｖａアプリケーションのためのオープンな実行時環境を提供する。ＪＶＭを保護し、ＪＶＭ自体を堅牢にするビルトインセキュリティはほとんどない。ＪＶＭ自体に添付するか、ＪＶＭを使用して攻撃を着手することは比較的ありふれたこと

である。したがって、Ｊａｖａアプリケーションコードに適用される保護の強度にかかわらず、ＪＶＭの脆弱性のため、ホワイトボックス攻撃を実行するために、ハッカーは常にＪＶＭを最弱のリンクとして使用し得る。より信頼度の高い、堅牢なＪＶＭなら、Ｊａｖａアプリケーションを保護し、ホワイトボックス攻撃を阻止する可能性はあるが、この場合、現在のＪａｖａセキュリティモデルに大きな変更が必要になり、これに関連してかなりの産業のサポートと順応が必要になる。したがって、アプリケーションをホワイトボックス環境で保護する工業規格のＪＶＭ内に信頼度の高い堅牢な要素を有することが好ましいと考えられる。

【先行技術文献】

【特許文献】

【００１３】

【特許文献１】２００９年３月１７日発行、「タンパリングに耐性のあるソフトウェア符号化および分析」と題するチョウ他の米国特許第７５０６１７７号（Patent No. 7,506,177 issued on 17 MAR 2009 to Chow et al. and titled TAMPER RESISTANT SOFTWARE ENCODING AND ANALYSIS）。

【特許文献２】２００８年１２月９日発行、「デジタルメディアを扱い配布するための安全な方法およびシステム」と題するジョンソン他の米国特許第７４６４２６９号（Patent No. 7,464,269 issued on 09 DEC 2008 to Johnson et al. and titled SECURE METHOD AND SYSTEM FOR HANDLING AND DISTRIBUTING DIGITAL MEDIA）。

【特許文献３】２００８年７月８日発行、「ホワイトボックス攻撃からコンピュータソフトウェアを保護するためのシステムおよび方法」と題するジョンソン他の米国特許７３９７９１６号（Patent No. 7,397,916 issued on 08 JUL 2008 to Johnson et al. and titled SYSTEM AND METHOD FOR PROTECTING COMPUTER SOFTWARE FROM A WHITE BOX ATTACK）。

【特許文献４】２００８年７月１日発行、「持続可能なデジタルウォーターマーキングのための方法およびシステム」と題するチョウ他の米国特許第７３９５４３３号（Patent No. 7,395,433 issued on 01 JUL 2008 to Chow et al. and titled METHOD AND SYSTEM FOR SUSTAINABLE DIGITAL WATERMARKING）。

【特許文献５】２００８年３月２５日発行、「タンパリング耐性のあるソフトウェアマスデータ符号化」と題するジョンソン他の米国特許第７３５００８５号（Patent No. 7,350,085 issued on 25 MAR 2008 to Johnson et al. and titled TAMPER RESISTANT SOFTWARE-MASS DATA ENCODING）。

【特許文献６】２００８年１月２９日発行、「セキュアなアクセスのための方法およびシステム」と題するチョウ他の米国特許第７３２５１４１号（Patent No. 7,325, 141 issued on 29 JAN 2008 to Chow et al. and titled METHOD AND SYSTEM FOR SECURE ACCESS）。

【特許文献７】２００５年１月１１日発行の「タンパリング耐性のあるソフトウェア符号化」と題するチョウ他の米国特許第６８４２８６２号（Patent No. 6,842,862 issued on １１ JAN 2005 to Chow et al. and titled TAMPER RESISTANT SOFTWARE ENCODING）。

【特許文献８】２００４年８月１７日発行、「タンパリング耐性のあるソフトウェア制御フロー符号化」と題するチョウ他の米国特許第６７７９１１４号（Patent No. 6,779, 114 issued on 17 AUG 2004 to Chow et al. and titled TAMPER RESISTANT SOFTWARE-CONTROL FLOW ENCODING）。

【特許文献９】２００３年７月１５日発行、「タンパリング耐性のあるソフトウェア符号化」と題するチョウの米国特許第６５９４７６１号（Patent No. 6,594,761 issued on 15 JUL 2003 to Chow et al. and titled TAMPER RESISTANT SOFTWARE ENCODING）。

【発明の概要】

【発明が解決しようとする課題】

【００１４】

本発明の目的は、これまでのＪａｖａプラットフォーム構成の主要な欠点を除去または

緩和することである。

【００１５】

本明細書では、Ｊａｖａプラットフォームの脆弱性に対処しかつ実行時のＪａｖａバイトコードを保護することができるセキュアなモジュールを提供する発明を開示する。セキュアなモジュールは、例としてＣ／Ｃ＋＋で実現されるものである。発明のセキュリティモジュールの構成がＣ／Ｃ＋＋によるものであるため、Ｃ／Ｃ＋＋ソフトウェアコードを安全にするセキュリティ技術を使用することができる。本発明の目的のための適切な技術は、カナダ、オンタリオ州、オタワ（Ｏｔｔａｗａ，Ｏｎｔａｒｉｏ，Ｃａｎａｄａ）のクロークウェア社（Ｃｌｏａｋｗａｒｅ　Ｉｎｃ）により提供されるものである。このように適切な目的のセキュリティ技術については、同一所有者の先行技術である特許文献１から９に詳しく記載されており、これらの特許の各々について、全文をここに引用により援用する。

【００１６】

上記の引用特許およびクロークウェア社からの関連製品により開示される既存のソフトウェアセキュリティ技術は、合法なアプリケーションに対するホワイトボックス攻撃を阻止するため、これらアプリケーションを、悪意ある（信頼性のない）実行環境で動作するアプリケーションの機能性と知的財産とともに保護するために使用される。これら既存のソフトウェアセキュリティ技術は、Ｃ／Ｃ＋＋のアプリケーションをネィティブな編集コードとともに保護し、伝統的なアプリケーション構築プロセスを強化することによってソフトウェアとセキュリティを不可分にする実用的なソースコードおよびバイナリ保護ツールを含む。

【課題を解決するための手段】

【００１７】

第１の実施例において、本発明は、Ｊａｖａバイトコードのタンパリング耐性を向上させるための装置を提供し、この装置は、構築時、Ｊａｖａバイトコードにセキュリティを適用するための保護ツールと、保護ツールから安全なＪａｖａバイトコードを受け、実行時に安全なＪａｖａバイトコードを起動するセキュリティモジュールと、保護ツールおよびセキュリティモジュールと統合された１以上の保護機構とを含み、１以上の保護機構が、Ｊａｖａバイトコードに対する静的および動的攻撃に対抗するよう動作する。

【００１８】

本発明のさらなる実施例において、この装置はまた被保護Ｊａｖａアプリケーションバイトコードスタブと、被保護アプリケーションペイロードと、暗号化されたクラスバイトコードフレームとを含む安全なＪａｖａバイトコードを含み、その各々が、構築時、保護ツールにより形成される。

【００１９】

本発明の他の実施例において、セキュリティモジュールが、Ｊａｖａヴァーチャルマシン環境への機能的拡張として独立に配布され、被保護Ｊａｖａアプリケーションの信頼の基礎を提供し、かつ安全なＪａｖａバイトコードが、ユーザの要望ごとに別々に配布される。

【００２０】

本発明の他の実施例において、保護ツールが、被保護アプリケーションペイロードが、被保護Ｊａｖａアプリケーションバイトコードスタブを介して起動されるよう命令する機構を含む。

【００２１】

本発明の他の実施例において、セキュリティモジュールが、被保護バイトコードクラスローダを含み、かつ保護ツールが、被保護アプリケーションペイロードが、被保護バイトコードクラスローダを用いて、暗号化されたクラスバイトコードフレームを介して起動されるよう命令する機構を含む。

【００２２】

他の実施例において、装置が、Ｃ、Ｃ＋＋、およびＪａｖａのうち１以上を含むプログ

ラミング言語で実現されるプログラミングエンジンから構成される。
【００２３】
他の実施例において、装置が、Ｊａｖａヴァーチャルマシンと対話することができるプログラミング言語で実現されるプログラミングエンジンから構成される。
【００２４】
他の実施例において、１以上の保護機構が、コンフィギュレーションオプションにより選択可能である。これらの保護機構が、保護ツールに形成される静的セキュリティハンドラおよびセキュリティモジュールで形成される動的セキュリティハンドラを含む。
【００２５】
他の実施例において、静的セキュリティハンドラが、ユーザからの暗号キーを含む暗号情報を受けるためのホワイトボックス（ＷＢ）静的セキュリティハンドラを含み、それにより、他の静的セキュリティハンドラの１以上により使用されるＷＢ暗号化キーデータならびに各々セキュリティモジュールの動的実行時保護の際に１以上の動的セキュリティハンドラにより使用されるＷＢ復号化キーデータおよびＷＢセキュリティモジュールユーティリティを生成する。
【００２６】
他の実施例において、静的セキュリティハンドラは、保護マーキング情報に応答して安全なＪａｖａバイトコードに対してハッシュコード保護を適用するためのバイトコード完全性検証（ＢＩＶ）静的セキュリティハンドラを含んでもよく、動的セキュリティハンドラが、実行時、ハッシュコード保護を検証するためのＢＩＶ動的セキュリティハンドラを含み、セキュリティモジュールが、検証失敗の際にタンパリングに対する対抗策を起動する。
【００２７】
他の実施例において、装置は、静的セキュリティハンドラを含み、その静的セキュリティハンドラが、構築時、被保護Ｊａｖａアプリケーションバイトコードスタブ、被保護アプリケーションペイロード、および暗号化クラスバイトコードフレームを形成するためのセキュアローディングバイトコード（ＳＬＢ）静的セキュリティハンドラを含んでもよく、かつ動的セキュリティハンドラが、メモリバッファ内へ、安全なＪａｖａアプリケーションバイトコードに対応する暗号化クラスバイトコードフレームをロードし、暗号化クラスに対応するＷＢ復号化キーデータを介して暗号化クラスバイトコードフレーム内に含まれる暗号化クラスの各々を復号化し、セキュリティモジュールクラスローダを介してアプリケーション作業空間内へ、各復号化されたクラスバイトコードをロードして、それによりアプリケーション作業空間内でＪａｖａアプリケーションバイトコードを実行するためのＳＬＢ動的セキュリティハンドラを含む。
【００２８】
本発明の他の局面および特徴については、以下の発明の特定の実施例の記載を添付の図面とともに読めば、当業者には明らかになるであろう。
【図面の簡単な説明】
【００２９】
本発明の実施例について添付の図面を参照して説明するが、説明は例示目的のみのものである。
【００３０】
【図１】ＪａｖａアプリケーションとネィティブコードをブリッジするＪＮＩの既知の概略を示す図である。
【００３１】
【図２】Ｊａｖａアプリケーションバイトコードに対する静的攻撃の既知の機構を示す図である。
【００３２】
【図３】Ｊａｖａアプリケーションバイトコードに対する動的攻撃の既知の機構を示す図である。

【００３３】
【図４】本発明によるＪａｖａバイトコード保護システムの概略を示す図である。
【００３４】
【図５】図４の上部に示す構築時プロセスの図であって、本発明による構築時のＪａｖａアプリケーションバイトコードを保護するプロセスを示す図である。
【００３５】
【図６】図４の下部に示す実行時プロセスの図であって、本発明による実行時のＪａｖａアプリケーションバイトコードを保護するプロセスを示す図である。
【００３６】
【図７】本発明によるスタートアップ時および実行時のアンチデバッグ能力を示す図である。
【００３７】
【図８】本発明による外部ホワイトボックス（ＷＢ）暗号ライブラリを示す図である。
【００３８】
【図９】本発明による内部ＷＢ暗号ファシリティを示す図である。
【００３９】
【図１０】本発明によるバイトコード保護ツールのプリプロセスを示す図である。
【００４０】
【図１１】本発明によるバイトコード完全性検証（ＢＩＶ）静的セキュリティハンドラの作業フローを示す図である。
【００４１】
【図１２】本発明によるＢＩＶ動的セキュリティハンドラの作業フローを示す図である。
【００４２】
【図１３】本発明によるセキュアローディングバイトコード（ＳＬＢ）静的セキュリティハンドラの作業フローを示す図である。
【００４３】
【図１４】本発明によるＳＬＢ動的セキュリティハンドラの作業フローを示す図である。
【００４４】
【図１５】本発明によるクロークされたＪａｖａアプリケーションのブートストラップインタフェースおよびセキュアなローディングを示す図である。
【００４５】
【図１６】本発明による動的バイトコード復号化（ＤＢＤ）静的セキュリティハンドラの作業フローを示す図である。
【００４６】
【図１７】本発明によるＤＢＤシーケンス図である。
【００４７】
【図１８】本発明によるＤＢＤ動的セキュリティハンドラの作業フローを示す図である。
【発明を実施するための形態】
【００４８】
図１からわかるとおり、Ｊａｖａプラットフォーム１００は、Ｊａｖａワールド（ＪＶＭ１０４、Ｊａｖａアプリケーション１０６およびＪＶＭ内にロードされるバイトコードのライブラリ１０８を含む）とネィティブコードワールド１１０（そのアプリケーションまたは共有ライブラリがＣ／Ｃ＋＋／アセンブラ等の他の言語で書かれかつホストＣＰＵＩＳＡにコンパイルされる）との間の双方向相互運用および対話をブリッジするためのファシリティを提供するＪａｖａネィティブインターフェース（ＪＮＩ）１０２をさらに含む。ＪａｖａプログラミングおよびＣ／Ｃ＋＋／アセンブラコードのＪＮＩアプリケーションプログラミングインターフェース（ＡＰＩ）を使用することにより、Ｃ／Ｃ＋＋／アセンブラネィティブバイナリコードがＪａｖａから呼び出し可能になりかつまたＪａｖａバイトコードを起動することができる。対話には二種類あり、すなわち、Ｊａｖａアプリケーションコードがネィティブメソッドを呼びだす場合の「ダウンコール」と、ネィ

ティブメソッドがデータにアクセスするかまたはＪＮＩ環境を介して所与のＪａｖａアプリケーションのメソッドを起動する場合の「アップコール」の２種類である。

【００４９】

実行時、本発明のセキュリティモジュールが、ＪＮＩを介してＪＶM内で同時に実行可能であり、それにより所与のＪａｖａアプリケーションが、発明のセキュリティモジュール内のセキュアなオペレーションを起動でき、このセキュアなオペレーションがＪａｖａアプリケーションおよびＪＶM内にロードされる他のＪａｖａライブラリコードにアクセスし、保護を実行することができるようになっている。

【００５０】

本発明のアプローチは、ＪＮＩ機構を介してＪＶM内で完全に保護され信頼されるセキュリティモジュールを導入することにより既存のＪＶMに対する効果的なセキュリティアドオンである。実行時、セキュリティモジュールは、信頼の基礎としてかつＪＶM内の保護「トランポリンおよびエンジン」として作用して、Ｊａｖａバイトコードの様々な保護を起動し実行する。このように、本発明は既存のＪａｖａプラットフォームに何らの広域な変更を要求するものではない。むしろ、既存と新しくデプロイされるシステムおよび装置の両方がこの解決法から直ちに恩恵を受けることができる。言い換えれば、本発明を、既存のＪａｖａインフラに対するセキュリティの拡張部分として扱い、現在のＪａｖａアプリケーションが直面するセキュリティの課題に対処することができる。こうして、本発明は、Ｊａｖａアプリケーションに対する静的および動的攻撃に応答して、実行時にバイトコードにアクセスしかつＪａｖａバイトコードの保護メソッドを実行する能力を活用するＪａｖａバイトコード保護セキュリティモジュールを提供する。

【００５１】

本発明は、信頼度の高い保護ツールおよびセキュリティモジュールをＪａｖａバイトコード保護システム内に提供する。本発明は、ＪＶMおよびＪａｖａセキュリティにのみに基づくＪａｖａアプリケーション保護には依存しない。むしろ、本発明は、ＪＮＩを介してＪＶMと協働することができる信頼されるゾーンであるＪａｖａバイトコード保護セキュリティモジュール（ＳＭ）を導入して、実行時にＪａｖａバイトコード保護を起動、実行および管理する。セキュリティモジュールの信頼性は既知の有効なセキュリティ保護を、そのプログラミング言語で保護ツールおよびセキュリティモジュールが書かれるＣ／Ｃ＋＋コードに適用ことにより保障される。この信頼度の高いＳＭでは、以下に説明するＳＭにより提供されるいくつかの保護により、信頼性はＳＭからＪａｖａアプリケーションおよびＪＶMにまで拡大される。

【００５２】

図２および図３は、Ｊａｖａバイトコードに対する典型的な静的および動的攻撃を示す。一般に、いずれか所与のＪａｖａアプリケーションをＪａｖａソースフォーム２０２で展開し、その後Ｊａｖａコンパイラ２０６によりＪａｖａバイトコード２０４にコンパイルするが、これは、配布前にアーカイバーユーティリティ２１０を用いて、アーカイブファイル２０８（すなわちＪＡＲファイル）上にストアされる。この配布は、コンパクトディスク（ＣＤ）等の媒体またはダウンロード可能なファイルを含む多くの形式で行われ得る。

【００５３】

静的攻撃者２１２は、通常、リバースエンジニアリングツール（Ｊａｖａデコンパイラ等）を使用して、配布媒体からのコードから価値のある知的財産情報２１４（すなわち財産価値のあるデータまたはソフトウェアアルゴリズム）を抽出する。その場合、攻撃者は、コードに違法な変更を加えるかさもなければ基礎となるコード２１６を傷つけ得る。所与のＪａｖａアプリケーション配布中に、Ｊａｖａバイトコードに対するこのような静的攻撃を阻止するため、本発明はアプリケーションバイトコードに対してあるレベルの保護を適用する。この保護は、配布の前に行われ、確実に静的攻撃が極めて困難な作業になるようにする。本発明により有効な保護が適用された後、アプリケーションバイトコード内に埋め込まれた知的財産は、簡単にリバースエンジニアされず、かつ保護されるバイトコ

ードのタンパリングは非実用的な行為となる。さらに、本発明の静的保護は、タンパリングされたバイトコードが、適正なＪＶＭによってロードされたり実行されたりできないという利点がある。

【００５４】

アプリケーションバイトコードに対する静的攻撃と違い、動的攻撃者３０２は、ＪＶＭがＪａｖａアプリケーションをロードおよび実行している間に動的攻撃ツールを用いてＪａｖａバイトコードに対して攻撃を実行する。動的攻撃ツールおよびメソッドを使用することにより、攻撃者はＪＶＭ３０４およびアプリケーションバイトコード３０６にアクセスができ、攻撃目的で、バイトコード３０８を観察かつ修正して直接的に元の指定された挙動および重要な値を理解または／および変更することができる。さらに、攻撃者は、元のバイトコード３０６のリフティングを含め、価値ある知的財産３１０およびバイトコードからの秘密を確かめることができる。実行時にＪａｖａコードに対するこのような動的攻撃を阻止するため、本発明は、配布前にアプリケーションのバイトコードに対して保護を構成しインプラントする。さらに、本発明は、実行時に、いかなる動的攻撃も必ず非現実的になるようにこれらの保護を実現する。本発明によれば、動的攻撃が防止されるのみならず、被保護アプリケーションに対して、動的攻撃を検出しこれを軽減する能力を付加し、またいかなる潜在的攻撃者にとってもこれらの攻撃が時間およびリソースの面から非常に費用のかかるものにする。

【００５５】

図４は、本発明によるバイトコード保護システム４００および関連のメソッドの概略図である。ここで、Ｊａｖａバイトコード保護システムは、２つの部分からなる。すなわち、構築時保護ツール４０２および実行時セキュリティモジュール４０４である。上記のとおり、バイトコード保護システムおよび関連のメソッドは、上記のカナダ、オンタリオ州、オタワのクロークウェア社から入手可能な技術を用いてＣ／Ｃ＋＋で実現される。

【００５６】

Ｊａｖａバイトコード保護ツール４０２を用いて、デプロイ前のＪａｖａバイトコード４０６にセキュリティ（すなわち「クローク」）を適用する。このＪａｖａバイトコード保護ツール４０２により、構築時に、セキュリティの設定と保護機構とを指定することができる。このツールは、元のＪａｖａアプリケーションのバイトコード４０６、セキュリティ仕様およびＷＢ暗号キーを入力として受け、Ｊａｖａバイトコード保護セキュリティモジュール４０４と関連して実行される「クロークされた」Ｊａｖａバイトコードを生成する。Ｊａｖａバイトコード保護ツール４０２は、バイトコードの起動法を指定する選択肢（被保護アプリケーションバイトコードスタブ（Ｐｒｏｔｅｃｔｅｄ　Ａｐｐｌｉｃａｔｉｏｎ　Ｂｙｔｅｃｏｄｅ　Ｓｔｕｂ）または被保護バイトコードクラスローダ（Ｐｒｏｔｅｃｔｅｄ　Ｂｙｅｔｃｏｄｅ　Ｃｌａｓｓ　Ｌｏａｄｅｒ）を介して等）およびデプロイ済の安全なＪａｖａバイトコード用セキュリティ技術を指定するための選択肢を含む。Ｊａｖａアプリケーションのクロークされたバイトコードは、２つにわけて配布される。すなわち、１）被保護Ｊａｖａアプリケーションバイトコードスタブ４０８（これは、標的ＪＶＭ環境４１０へロードされる）および２）被保護データファイル４１２とホワイトボックスセキュリティモジュール（ＷＢＳＭ）ユーティリティ４１４であり、これらはロードされ、実行時に別々にＳＭによりアクセスされる。発明のＪａｖａバイトコード保護セキュリティモジュール４０４は、アプリケーションの準備アプローチの段階まで、これら２つの部分とともにまたは単独で配布することができる。一般に、セキュリティモジュール４０４は、一度インストールするといずれのＪａｖａアプリケーションのクロークされたバイトコードにも適用できると言う点で汎用である。

【００５７】

本件のＪａｖａバイトコード保護システム４００と関連して使用する、発明のバイトコード保護のための様々なメソッドが有効である。このバイトコード保護メソッドは、各々、バイトコード形式のＪａｖａアプリケーションに対する静的および動的攻撃に対処する。

【００５８】
バイトコード保護の一メソッドは、ホワイトボックス暗号または「ＷＢ暗号」を含み、これは、これらのオペレーションが、暗号キーおよび他の暗号値を漏洩せずに悪意ある環境内で実行できるよう、暗号アルゴリズムを保護する独自の暗号技術である。言い換えれば、ＷＢ暗号法は、直接攻撃に対しても実行可能である。本発明は、外部ＷＢ暗号ライブラリおよび内部ＷＢ暗号ファシリティを含む２種類のＷＢ暗号技術を取り入れる。
【００５９】
外部ＷＢ暗号ライブラリは、Ｃで実現し、かつＷＢ暗号オペレーションを被保護Ｊａｖａアプリケーションが使用でき、かつキーを含む価値のある情報を全く解放することなく実行できるように、隠し暗号キーおよび他の暗号情報による耐タンパリング特性で保護される。本発明の内部ＷＢ暗号ファシリティは、暗号情報およびキーを受け入れる構築時保護ツール４０２の機能要素であり、発明の保護ツールおよびセキュリティモジュール４０４が様々な形式のＪａｖａアプリケーションのバイトコードおよび関連情報を暗号化しかつ復号化するために使用するＷＢキーデータおよびにユーティリティを生成する。
【００６０】
発明によるバイトコード保護の他のメソッドは、バイトコード完全性検証（ＢＩＶ、Ｂｙｔｏｃｏｄｅ Ｉｎｔｅｇｒｉｔｙ Ｖｅｒｉｆｉｃａｔｉｏｎ）を含む。ＢＩＶを介する保護により、Ｊａｖａクラスまたはメソッドのコードに対する静的および動的タンパリング攻撃を検出しかつ弱めることができる一方で、クラスまたは実行されるＪａｖａメソッドをロードすることができる。構築時、本発明のメソッドは、元のアプリケーションのアーカイブファイルからＪＡＲファイル、クラスおよびメソッドのバイトコードの静的ハッシュ値を計算する。ロードおよび実行時、発明のメソッドは、ＪＶＭ４１０でロードされるクラスおよびメソッドのバイトコードにアドレスすることにより動的ハッシュ値を計算しかつ動的ハッシュ値を静的ハッシュ値に対して照合することで完全性の検証を行う。
【００６１】
発明によるバイトコード保護の他のメソッドは、アンチデバッグ（ＡＤ、Ａｎｔｉ－Ｄｅｂｕｇ）を含み、これについては、図７を参照して後述する。ＡＤは、図４に示す動的セキュリティハンドラ４１６の１つである。ＡＤ保護により実行時にデバッガを使用する動的攻撃を阻止しかつ検出することができる。ＡＤは、スタートアップおよび実行時のシステム環境の内部および外部状態をモニタすることによって攻撃を検出する技術から構成される。アンチデバッグ攻撃が検出されると適切な対抗手段が起動される。
【００６２】
発明によるバイトコード保護の他のメソッドは、セキュアローディングバイトコード（ＳＬＢ，Ｓｅｃｕｒｅ Ｌｏａｄｉｎｇ Ｂｙｔｅｃｏｄｅ）である。このＳＬＢ保護法は、ＪＶＭ４１０にロードされる前のアーカイブファイルおよびＪａｖａクラスコードに対する静的リバースエンジニアおよびタンパリング攻撃を阻止しかつ検出する。構築時、ＳＬＢ保護法は、もとのアプリケーションアーカイブファイルからのＪＡＲファイルおよび選択されたクラスファイルを暗号化しアプリケーションスタブクラスを導入する。ＪＶＭ４１０が被保護アプリケーションをロードする時、ＪＶＭ４１０は、まずアプリケーションスタブクラスをロードし、次に被保護アプリケーションのロードをトリガする。以下でさらに説明するＳＬＢ動的セキュリティハンドラ４１６は、実行時実行の際にＪＮＩ４１８を介してＪＶＭ４１０と接続する発明のＪａｖａバイトコード保護セキュリティモジュール４０４の機能要素である。ＳＬＢ動的セキュリティハンドラ４１６は、ＪＶＭ４１０の作業空間への被保護Ｊａｖａアプリケーションバイトコードのロードを管理かつ制御する。
【００６３】
発明によるバイトコード保護の他のメソッドは、動的バイトコード復号化（ＤＢＤ，Ｄｙｎａｍｉｃ Ｂｙｔｅｃｏｄｅ Ｄｅｃｒｙｐｔｉｏｎ）である。ＤＢＤ保護法により実行時のＪａｖａクラスまたはメソッドのコードに対する動的攻撃が阻止および軽減され

る。
【００６４】
発明によるバイトコード保護の他のメソッドは、転送実行および部分実行の両方を含む。これらの保護法は両方とも元の実行の一部をセキュリティモジュール４０４へ移動させ、ＪＶＭ４１０内で実行の一部のみが晒されて、確実に実行時の動的コードリフティング攻撃の阻止および軽減を行う。たとえば、あるＪａｖａバイトコードは、セキュリティモジュール４０４内で保護実行できるＣコード（Ｊ２Ｃ）に変換することができる。
【００６５】
発明による他のバイトコード保護メソッドは、バイトコード変換を含む。この種の保護は、データフロー変換および制御フロー変換を含む技術により達成できる。バイトコード変換は、元の機能性を維持したまま、もとのバイトコードを別のコード構造に変換することができる。変換されたバイトコードは、リバースエンジニアすることがより難しくかつタンパリング耐性を有する。
【００６６】
図５を参照して、Ｊａｖａバイトコード保護ツール４０２は、元のアプリケーションのバイトコードに対して様々な保護技術を適用する。Ｊａｖａバイトコード保護ツール４０２はこうして、保護されたバイトコードおよび関連のデータならびに実行時にＪａｖａバイトコードセキュリティモジュールとともに作用して、Ｊａｖａバイトコードに対するこれら指定された保護技術を適用するユーティリティを生成する。Ｊａｖａバイトコード保護ツール４０２は、ユーザインタフェース５１８を介して暗号情報およびキー５０２、元のＪＡＲファイル５０４ならびにコンフィギュレーションオプション５０６の３つの入力を受け、この３種類のオペレーションを実行する。
【００６７】
第１の基本的オペレーションは、ＷＢキーデータおよびユーティリティの生成を含む。暗号情報およびキー５０２を使用して、ＷＢ静的ハンドラ５０８は、様々な静的セキュリティハンドラ（それぞれ詳細については後述）およびツール自体により使用されるＷＢ暗号化キーデータ５１０を生成する。また、保護ツール４０２の構築時プロセスにより、データ保護フォルダ５１４における実行時データ５１２の一部として記憶されるＷＢ復号化キーデータが生成される。ＷＢセキュリティモジュール（ＳＭ）ユーティリティ５１６が提供され、ＷＢ復号化キーデータを使用して実行時に動的セキュリティハンドラにより起動されるＷＢ復号化オペレーションが実行される。
【００６８】
第２の基本オペレーションは、保護技術の適用を含む。コンフィギュレーションオプションによれば、Ｊａｖａバイトコード保護ツール４０２は、様々な静的セキュリティハンドラを適用して、アプリケーションバイトコードを元の形式から保護された形式に修正する。そうする上で、このオペレーションは、被保護Ｊａｖａアプリケーションバイトコードスタブ４０８と、様々な保護形式で保護されたアプリケーションバイトコードおよび重要な実行時データを含む関連保護データファイルを生成する。
【００６９】
第３の基本的オペレーションは、被保護Ｊａｖａバイトコードのデプロイ可能な形態のパッケージングを含む。プロセスの終了時に、Ｊａｖａアプリケーションバイトコードスタブ４０８がＪＶＭ４１０によりロードできるように、Ｊａｖａバイトコード保護ツール４０２は、適正にすべての出力ファイルを構築かつパックする。このＪａｖａアプリケーションバイトコードスタブ４０８は、クロークされたＪａｖａアプリケーション起動へのエントリポイントであり、様々な形式を取りうる。すなわち、その中には、外部プログラムにより起動可能なクラスファイル、他のＪａｖａクラスによって起動されるクラスファイル、またはＪａｖａクラスローダが含まれる。Ｊａｖａバイトコード保護ツール４０２も、ＷＢＳＭユーティリティ５１６が、Ｊａｖａバイトコードセキュリティモジュールの機能要素により起動され得るように、すべての出力ファイルを適正に構築しパックする。さらに、Ｊａｖａバイトコード保護ツール４０２も、すべての保護データファイルを、Ｊ

ａｖａバイトコードセキュリティモジュール４０４のいくつかの機能要素によりによりアクセスできるように、すべての出力ファイルを適正に構築しかつパックする。

【００７０】

図５は、Ｊａｖａアプリケーションバイトコードを保護する上記の構築時プロセスの概略図である。図５に関連して、主な機能要素およびデータファイルについてここで説明する。

【００７１】

Ｊａｖａバイトコード保護ツール４０２は、ユーザコマンドおよび主な入力を受けるため、ユーザとのインタフェースをとるためのユーザインタフェース５１８を含む。コマンドおよび入力には、暗号アルゴリズムセレクションおよび元のキー材料を含む暗号情報およびキー５０２、保護対象の非保護バイトコードを含む元のアプリケーションバイトコードアーカイブファイル５０４およびたとえばユーザが特定のＪａｖａクラスおよびメソッドを保護対象にするか否か指定できる等、アプリケーションバイトコードを保護するため、何に対しかつどのようにＪａｖａバイトコード保護ツール４０２を運用するかというユーザのオプションを含むコンフィギュレーションオプション５０６を含む。

【００７２】

Ｊａｖａバイトコード保護ツール４０２も、保護マネジャ５２０を含む。保護マネジャ５２０は、コンフィギュレーションオプション５０６を解釈し、様々な保護技術を依存順位でコーディネートして、それらを組み合わせ、結果として得られる全体的保護が、各個別の保護より強化されたものになるように設けられる。また、マネジャ５２０は、Ｊａｖａバイトコード保護ツール４０２の他の機能要素により共通に使用されるユーティリティを含む。

【００７３】

Ｊａｖａバイトコード保護ツール４０２も、静的セキュリティハンドラ５２２を含む。各個々の静的ハンドラは、保護マネジャ５２０により起動されて、それぞれ予め定められた保護技術を実行する。実施例では、ＷＢ静的ハンドラ５０８、ＢＩＶ静的ハンドラ５２４、ＡＤ静的ハンドラ５２６、ＳＬＢ静的ハンドラ５２８、ＤＢＤ静的ハンドラ５３０、転送実行静的ハンドラ５３２、部分実行静的ハンドラ５３４、コード変換ツール５３６を示す。各静的セキュリティハンドラについては、以下の項でより詳細に説明する。保護マネジャ５２０および静的セキュリティハンドラ５２２は、それらが協働して、容易に更なる新しいセキュリティハンドラと保護ツールを統合することにより、セキュリティ能力および新たな保護を付加かつ拡張するプラグイン機構を提供するよう設計されている。

【００７４】

Ｊａｖａバイトコード保護ツール４０２も、ＷＢ静的セキュリティハンドラ５０８が生成するＷＢ暗号化キーデータ５１０を含む。ＷＢ暗号キーデータ５１０は、マネジャ５２０および静的セキュリティハンドラ５２２により使用され、いくつかの形式のバイトコードおよび保護データを暗号化する。

【００７５】

Ｊａｖａバイトコード保護ツール４０２はまたＷＢ静的セキュリティハンドラ５１０が生成するＷＢＳＭユーティリティ５１６を含む。ＷＢＳＭユーティリティ５１６は、セキュリティモジュール４０４内で、動的セキュリティハンドラ（後に詳説）により使用される。

【００７６】

Ｊａｖａバイトコード保護ツール４０２はまた被保護Ｊａｖａアプリケーションバイトコードスタブ４０８を含む。スタブ４０８は、セキュアなバイトコードローダ機能をまずロードし次にトリガして、保護データファイルから真の被保護バイトコードをロードするためのＪＶＭ４１０用被保護Ｊａｖａアプリケーションのブートストラップのみを含む。

【００７７】

Ｊａｖａバイトコード保護ツール４０２はまたツールが生成する被保護Ｊ２Ｃライブラリ５３８を含む。被保護Ｊ２Ｃライブラリ５３８は、Ｊａｖａバイトコードから変換された

、Ｃの様々な被保護コードを含む。このライブライリは、Ｊａｖａバイトコードセキュリティモジュールにより動的にリンクされ起動される。

【００７８】

Ｊａｖａバイトコード保護ツール４０２はまた被保護バイトコードデータ５４０を含む。この被保護バイトコードデータ５４０は、ツールが生成する一種の保護データファイルであり、様々な被保護バイトコードを含む。

【００７９】

Ｊａｖａバイトコード保護ツール４０２はまた実行時データ５１２を含む。この実行時データ５１２は、これに限定されるわけではないが、ＷＢ復号化キーデータ、完全性検証静的ハッシュ値、被保護クラスおよびメソッドの情報ならびに表等の各種セキュリティ関連情報を含む。

【００８０】

なお、Ｊａｖａバイトコード保護ツール４０２は、ダウンロード性を示す。このように、この保護ツールからのすべての出力（被保護Ｊａｖａアプリケーションバイトコードスタブ４０８、被保護Ｊ２Ｃライブラリ５３８、被保護バイトコードデータ５４０および実行時データ５１２を含む）は、実行時にダウンロード可能である。

【００８１】

図６は、図４に示すＪａｖａバイトコード保護セキュリティモジュール４０４に関する、本発明によるＪａｖａバイトコードを保護する実行時プロセスを示す概略図である。上記のとおり、Ｊａｖａバイトコード保護セキュリティモジュール４０４は、Ｃプログラミング言語で展開し、それ自体はカナダ、オンタリオ州、オタワのクロークウェア社が提供するような、耐タンパリング技術により保護され、堅牢でタンパリング耐性がある。なお、Ｊａｖａバイトコード保護ツール４０２およびセキュリティモジュール４０４の基礎となるプログラミングエンジンは、他のプログラミング言語で展開されるエンジンでもよい。実際、この発明の基礎となるセキュリティモジュール４０４は、その言語がＪａｖａヴァーチャルマシン（Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ）とインタフェースできる限り、他の言語で展開することができる。

【００８２】

実行時を開始する際に、ＪＶＭ４１０が、通常のＪａｖａアプリケーションをロードする際と同様、被保護Ｊａｖａアプリケーションバイトコードスタブ４０８をロードする。これにより、Ｊａｖａアプリケーションバイトコードスタブ４０８がトリガされて、ＪＮＩ６０２を介してセキュリティモジュール４０４とインタフェースをとることにより信用できかつ保護されるＪａｖａアプリケーションバイトコードをブートストラップする。実行時、セキュリティモジュール４０４は、Ｊａｖａアプリケーションバイトコードおよび実行を確保しかつ保護し、それによりバイトコードおよび実行に対する動的攻撃を阻止するように、データフローを管理制御する役割をする。

【００８３】

さらに図６を参照して、ここで、主な機能要素およびデータファイルについて説明する。セキュリティモジュール４０４へ／からのデータおよびフローの制御は、Ｊａｖａアプリケーションバイトコード作業空間６０４を介して行われる。作業空間６０４は、ＪＶＭ４１０内のＪａｖａアプリケーションのための仮想作業空間である。アプリケーションをロードしかつ実行することを含む実行時の様々な状態で、ＪＶＭ４１０に存在する実際のアプリケーションバイトコードは、様々に管理される。作業空間の各状態には、合法で完全に機能的だが、完璧ではないアプリケーションバイトコードを含む。随意には、これらバイトコードのいくつかの部分は、ＪａｖａおよびＣ実行オプションで転送実行を有効にする等の構築時のコンフィギュレーション設定によって保護された状態に常に保つことができる。バイトコードのこの部分を実行する必要があれば、セキュリティモジュール４０４は、作業空間内のこれらをジャストインタイムで、ＪＶＭ４１０内へロードし復元し、実行が終わればこれらを除去する。また、いくつかの元のメソッドバイトコードが、ＪＶＭ４１０からは直接的に可視的でないＣ機能に変換されており、セキュリティモジュール

４０４によってのみ起動することができる。このメソッドでは、攻撃者は、実行時、どの瞬間においても、元のアプリケーションバイトコードの部分しか見ることができず、アプリケーションバイトコード全体をリバースエンジニアすることは極めて難しくなる。

【００８４】

セキュリティモジュール４０４（ＳＭ）は、また、ＪＮＩＳＭブリッジ４１８と呼ぶ図６に示すブリッジ機構を含む。ＪＮＩＳＭブリッジ機構４１８は、ＪＶＭ４１０とセキュリティモジュール４０４との間でＪＮＩ６０２を介して接続および相互機能を可能にする対話要素である。ＪＮＩＳＭブリッジ４１８のサブ要素は、ＪＶＭ４１０とネィティブコードとの間に唯一の対話機構を提供するＪＮＩ６０２を含む。また、サブ要素には、ダウンコールスタブ６１２およびアップコールスタブ６０８が含まれる。これらのスタブは、ＪＶＭ４１０のＪａｖａアプリケーションバイトコード作業空間６０４からのダウンコールを、ネィティブプログラミングコードのセキュリティモジュール４０４を介して動的セキュリティハンドラ６１１へリディレクトしかつセキュリティモジュール４０４からのアップコールをＪＶＭ６１０へりディレクトするアプリケーションプログラミングインターフェースを提供する。図示の第３のサブ要素は、ＳＭマネジャ６１０である。ＳＭマネジャ６１０は、セキュリティモジュール４０４のためのコントローラでありかつコーディネータである。Ｊａｖａアプリケーションバイトコードに対する各種の指定された保護を管理維持するのみならず、セキュリティモジュール４０４自体についても管理維持を行う。また、セキュリティモジュール４０４の他の機能要素により共通に使用されるユーティリティも含む。

【００８５】

セキュリティモジュール４０４は、複数の動的セキュリティハンドラ６１１を含む。各個別の動的セキュリティハンドラ６１１を起動して独自の保護技術を実行する。図示の通り、いくつかの実施例による動的セキュリティハンドラは、ＷＢ動的セキュリティハンドラ６１４と、バイトコード完全性検証動的セキュリティハンドラ６１６と、アンチデバッグ動的セキュリティハンドラ６１８と、ＳＬＢ動的セキュリティハンドラ６２０と、ＤＢＤ動的セキュリティハンドラ６２２と、転送実行動的セキュリティハンドラ６２４と、部分実行動的セキュリティハンドラ６２６と、コード変換６２８とを含む。動的セキュリティハンドラ６１１の詳細については、後述する。

【００８６】

構築時Ｊａｖａバイトコード保護ツール４０２によるコーディネートで、ＳＭマネジャ６１０および動的セキュリティハンドラ６１１は、協働して、セキュリティモジュールと追加の新たな動的セキュリティハンドラを容易に統合することによりセキュリティ能力および新たな保護を追加拡張するプラグイン機構を提供するよう設計される。

【００８７】

図７において、外部アンチデバッグモニタリングのための発明のメソッドの実施例を示す。ここで、Ｊａｖａプラットフォームデバッグアーキテクチャ（ＪＰＤＡ、Ｊａｖａ Ｐｌａｔｆｏｒｍ Ｄｅｂｕｇ Ａｒｃｈｉｔｅｃｔｕｒｅ）によりＪａｖａアプリケーションのデバッグ能力を促進する。発明のメソッドは、ＪＰＤＡに基づくデバッグの有効化およびそれに続くデバッグ活動の検知に焦点を当てる。図示の通り、多層による防御戦略を用いて、実行中のＪＶＭプロセス内での静的および動的デバッグ活動を捕捉する可能性を最大限にする。図７に示すＡＤのメソッドにおける３つのエージェントは、通常のまたは合法のデバッグ活動が実行できるよう構成することができる。３つのエージェントは、カーネルモニタエージェント（ＫＭＡ、Ｋｅｒｎｅｌ Ｍｏｎｉｔｏｒ Ａｇｅｎｔ）７０２、デバッガアタッチメントモニタエージェント（ＤＡＭＡ、Ｄｅｂｕｇｇｅｒ Ａｔｔａｃｈｍｅｎｔ Ａｇｅｎｔ）７１０およびデバッギングプロシージャモニタエージェント（ＤＰＭＡ、Ｄｅｂｕｇｇｉｎｇ Ｐｒｏｃｅｄｕｒｅ Ｍｏｎｉｔｏｒ Ａｇｅｎｔ）７１８を含む。

【００８８】

カーネルスペース７０１にアクセスするＫＭＡ７０２に関して、ＪＶＭプロセスが、な

んらかのデバッグ機能が実行できる前にそのメモリ空間にデバッギングライブラリ７０５をロードすることが必要である。ＫＭＡ７０２は、Ｊａｖａアプリケーション開始時に生成される。ＫＭＡ７０２は、カーネルからのそれ自身のプロセスマップ７０３を定期的にチェックして、ＪＤＰＡに関連するライブラリがそのメモリ空間にロードされるかどうか決定する。これらのライブラリが見つかれば、適正な関連動作を行う。

【００８９】

ＤＡＭＡ７１０に関連して、このエージェントは、防御の第２のラインの役割をする。ＤＡＭＡ７１０は、Ｊａｖａヴァーチャルマシーンツールインターフェース（ＪＶＭＴＩ、Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ　Ｔｏｏｌ　Ｉｎｔｅｒｆａｃｅ）能力で促進されかつＪＶＭが７００を開始するときにロードされる。コールバック機能を設けて、実行時に創出されるすべてのスレッドについて定常的にスレッド開始スクリーンをモニタする。デバッギングを行うのに必須と考えられるいくつかのスレッドをＪＶＭがロードするたびに、Ｊａｖａアプリケーションにおける添付されたＪＤＰＡデバッガの活動を捕捉することができる。これに関して、ＡＭＡは、スレッド開始リスナ７０７を有効化し、新たなスレッド開始７０９を検出しかつＪＤＰＡ関連のスレッド７１１を検出する。

【００９０】

ＤＰＭＡ７１８に関しては、このエージェントは、防護の第３のラインとして設けられる。ＤＰＭＡ７１８も、ＪＶＭＴＩ環境下で動作する。デバッギングプロシージャ（ブレークポイントラインにヒットする等）を監視するコールバック機能は、そのような行為があるたびにトリガされる。スレッドとブレーキングポイントの位置等の詳細なメッセージを収集することができる。これに関して、ＤＭＰＡは、ラインブレークリスナ７１３を有効化し、デバッギング動作７１５を検出して、なんらかのスレッドおよびメソッド情報７１７をレポートする。ＫＭＡ、ＤＡＭＡおよびＤＰＭＡの各々が、動作をトリガでき、ＪＶＭ７２６を無効化する。

【００９１】

上記の静的および動的セキュリティハンドラについてここで詳細に説明する。ＷＢセキュリティハンドラは、図８に示す外部ＷＢ暗号ライブラリおよび図９に示す内部ＷＢ暗号ファシリティを含む。

【００９２】

ＷＢ動的セキュリティハンドラ６１４が提供する図８の外部ＷＢ暗号ライブラリは、ＪＮＩセキュリティモジュールインターフェース８０４を介してＷＢ暗号化および復号化のためのＪａｖａアプリケーションにより使用されるライブラリを提供する。ＷＢ静的ハンドラ５０８は、ユーザから暗号情報およびオリジナルキー５０２を受けかつ必要に応じて配布かつロールが可能なＷＢキーデータ８０３を生成し、これを暗号ライブラリがセキュアな暗号オペレーションのために使用できる。

【００９３】

内部ＷＢ暗号ファシリティは、ＷＢ静的ハンドラ５０８を含み、いくつかの静的および動的要素について図９に示す。ＷＢ静的ハンドラ５０８は、ユーザからの暗号情報およびオリジナルキー５０２を受け、かつＷＢ暗号化キーデータ９０４を生成するが、これを他の静的セキュリティハンドラ９０６が、様々な保護技術の一部として様々な形式のアプリケーションバイトコードに対する暗号化オペレーションのために使用する。ＷＢ静的ハンドラ５０８はまたＷＢ復号キーデータ９０８を生成して、各々動的セキュリティハンドラ６１１が使用するＷＢセキュリティモジュールユーティリティ６３０を提供して、セキュリティモジュール９１４が動的保護を実行する一方で復号化オペレーションを実行する。

【００９４】

Ｊａｖａバイトコード保護ツール４０２はまた図１０に示す前処理メソッドを含む。前処理ツール１００１は、もとのＪａｖａアプリケーションバイトコードアーカイブファイル１００５を受け、これらをもとのアプリケーションバイトコードのインターナルプレゼンテーション（ＩＲ、Ｉｎｔｅｒｎａｌ　Ｒｅｐｒｅｓｅｎｔａｔｉｏｎ）に翻訳する。特定のクラスおよびメソッドについては、その後、保護およびユーザオプション１００３

によるそれらの保護の態様についてマークする。こうして保護マーク情報１００４が生成される。ＩＲ形式のもとのアプリケーションバイトコード１０００および保護マーク情報１００４は、必要な保護のために各々静的セキュリティハンドラ５２２により使用される。

【００９５】

Ｊａｖａ保護ツール４０２の各静的セキュリティハンドラ内には、バイトコード完全性検証（ＢＩＶ）が存在する。図１１は、ＢＩＶ静的セキュリティハンドラ５２４の作業フローを示す図である。また、図１２は、ＢＩＶ動的セキュリティハンドラ６１６の作業フローを示す図である。ここで、ＢＩＶは、実行時にＪａｖａバイトコードの動的完全性検証能力を導入することにより独自のタンパリング耐性保護を提供する。一般に、構築時、ツールを使って、ＢＩＶデータ１２０２が生成されその後保護されるＢＩＶ保護を必要とするクラスおよびメソッドに署名し、かつＢＩＶアクションがＪａｖａバイトコードに構築される。実行時、ＢＩＶアクションは、それぞれのバイトコードについて、動的セキュアハッシュ値がジャストインタイムで計算されるＢＩＶ被保護クラスおよびメソッドのためのＪａｖａバイトコード保護セキュリティモジュール４０４を介してトリガされる。静的および動的セキュアなハッシュ値は、セキュアな形式で表現され、タンパレジスタンスゲートキーパー（ＴＲＧＫ、Ｔａｍｐｅｒ　Ｒｅｓｉｓｔａｎｃｅ　Ｇａｔｅ　Ｋｅｅｐｅｒ）１２１６へ成功および／または失敗コールバック機能をフィードする。ＴＲＧＫ１２１６は、静的セキュアハッシュ値と動的セキュアハッシュ値を明示的に比較せずに、ＢＩＶチェックが成功したか否かを判別する。これは、特別に設計された数学的計算の形で適切なアルゴリズムを介して行うことができる。静的セキュアハッシュ値および動的セキュアハッシュ値が同じの場合は、一般にＢＩＶチェックが通りかつ成功のコールバック機能が起動され得ることを示す。そうでない場合、静的セキュアハッシュ値および動的セキュアハッシュ値が同じでない場合は、特定のクラスまたはメソッドのタンパリングが検出され、ＢＩＶチェックが失敗であることを示す。こうして、失敗のコールバック機能を起動することができる。これらコールバック機能は、検出されるタンパリング攻撃に対するユーザ定義の対抗手段である。

【００９６】

本発明において、Ｊａｖａバイトコードの動的セキュアハッシュ値１２１４を計算するプロセスは、計算が、実行可能なものに割り当てられたメモリから直接ネィティブコードを選ぶに過ぎない、通常のネーティブバイナリコードに対する計算の典型的処理とは異なる。通常、アプリケーションコードは、Ｊａｖａ実行時にはメモリから直接コードセグメントを得ることはできない。その代り、アプリケーションコードは、ＪＶＭ４１０機構を通じてクラスまたはメソッドのバイトコードを入手する。この発明にでは、ＪＶＭ４１０へのアップコールを使用してバイトコードを回収し、その後セキュアな動的ハッシュ値１２１４を計算しかつ回収したバイトコードを予め登録されているハッシュ値と照合して完全性検証チェックを行うことにより、セキュリティモジュールが、この能力およびＪＮＩインタフェースを強化する。

【００９７】

図１１に関連して、バイトコード完全性検証静的セキュリティハンドラ５２４は、バイトコード署名を含むことがわかる。ＢＩＶ静的セキュリティハンドラ５２４の主要な機能の１つは、アプリケーションバイトコードをウォークスルーして保護マーク情報１１０５を使用してクラスおよびメソッドの各々をチェックして、どのクラスまたはメソッドがＢＩＶ保護を必要とするかを判別することである。あるクラスまたはメソッドがＢＩＶ保護を必要とする場合、特定のハッシュ値を、セキュアなハッシュ計算をその特定のクラスまたはメソッドのバイトコード１１０６または１１０７に適用することにより計算する。一般には、計算技術において知られるセキュアなハッシュ計算アルゴリズムを使用する。これら結果として得られるハッシュ値は、実行時に効果的に使用できるよう組織化され構造化されたメソッドで、ＢＩＶデータ１１０８として記憶される。

【００９８】

バイトコード完全性検証静的セキュリティハンドラのＢＩＶデータ１１０８は、クラスおよびメソッドの静的ハッシュ値のデータおよびＷＢＢＩＶ暗号キーデータ等の他の情報を含むデータ容器である。これらデータは、動的ＢＩＶセキュリティハンドラにより実行時に使用される。より効果的に使用するために、ＢＩＶデータ１１０８は、保護対象のクラスおよびメソッドの各々についての対応の情報およびそれらの静的ハッシュ値で構成される。

【００９９】

バイトコード完全性検証静的セキュリティハンドラ５２４はまたＢＩＶデータ１１０８を変換および暗号化する役割をする。ＢＩＶデータ１１０８の完全性を維持することは大変重要である。ＢＩＶデータは、ネットワーク経由で転送またはダウンロード可能である。したがって、本発明はこれらに対して、使用のためのパッキングの一部として変換と暗号化とを適用する。このような保護がなければ、ＢＩＶデータのタンパリングは、ＢＩＶ保護を破壊するステップとなり得る。パッケージの際、ＢＩＶ静的セキュリティハンドラは、敏感なＢＩＶデータに対する静的攻撃を防ぐようにＢＩＶデータに対する二重の保護を行う。まず、静的ハッシュ値を実行時に動的ＢＩＶセキュリティハンドラ６１６で変換された形式で演算できるように、ＢＩＶ静的セキュリティハンドラ５２４がこれら値に対するデータの変換を行う。これにより、確実に実際の単純な値が暴露されないようになる。次に、ＢＩＶ静的セキュリティハンドラ５２４は、これら変換された値の暗号化を行い、動的に使用される前にこれらの変換された値になんらのタンパリングも発生しないようにする。

【０１００】

なお、ＢＩＶデータ１１０８は、動的セキュリティハンドラ６１１により実行時に使用される一種の実行時データである。実行時データは、ユーザオプションにより単一のファイルまたは複数のファイルに組織化され記憶される。複数の実行時データファイル形式には、より細かい粒度でデータ情報をアップデートおよびダウンロードできるなどいくつか利点がある。たとえば、ＢＩＶデータ１１０８が、保護対象のＪａｖａクラスの各々について構成され得る。このように、ＢＩＶ保護は、より実際的にクラスごとに行うことができる。

【０１０１】

バイトコード完全性検証静的セキュリティハンドラ５２４は、また、独自のＢＩＶトリガを提供する。実行時トリガＢＩＶへの２つのアプローチは、外部ＢＩＶＡＰＩおよび内部ＢＩＶトリガを介して提供される。本発明のシステムの一部である第１のアプローチとして、いくつかの外部ＢＩＶＡＰＩが提供され、ユーザは、ＢＩＶチェックを行う明確なアイディアを有する適切な場所でそれらを使用する。ユーザは、どのＪａｖａクラスまたはメソッドがＢＩＶチェックを必要とするか示すことができる。ユーザは、コールバック機能を使用することにより軽減アクションの完全なコントロールを有する。他のメソッドは、ユーザが外部のＡＰＩを起動することによりトリガすることへの代替メソッドである。その代りに、ＢＩＶトリガをＪａｖａバイトコード保護セキュリティモジュールのいくつかの機能内に予め構築することができる。Ｊａｖａアプリケーションがこれらの機能を起動するたびに、予め決められ態様で、内部のＢＩＶアクションがトリガされる。いくつかの軽減アクションは、予め規定されており、セキュリティモジュールにより内部に取り込まれる。しかしながら、ユーザは依然として、軽減アクションに対する部分的コントロールを有する。これは設定に従って行動するよう、予め設定されたＡＰＩを提供して、セキュリティモジュールが取る軽減アクションを予めユーザに設定してもらうことにより有効となる。一般に、ユーザは、外部ＢＩＶＡＰＩを使用するかどうかおよび使用する場所に対して完全なコントロールを有し、かつ構築時内部ＢＩＶトリガを使用するかどうかについて間接的コントロールを有する。セキュリティモジュールにより隠されかつ制御されるので、内部ＢＩＶをトリガする場所に対しては、ユーザはなんらのコントロールも持たない。

【０１０２】

図１２に関して、バイトコード完全性検証動的セキュリティハンドラ６１６が、ＢＩＶ初期化を含むことがわかる。ＢＩＶ初期化が行われて、ＷＢＢＩＶ復号化キーデータ１２０３を使用してセキュアな静的ＢＩＶデータ１２０２がロードされ復号化され、その後、これらはセキュアな形式でメモリ内にロードされる。ＢＩＶ初期化を２つのメソッドで実行することができる。すなわち、セキュリティモジュールの初期化の一部としてまたは動的ＢＩＶの際のオンデマンドである。第１のメソッドについては、被保護Ｊａｖａアプリケーションをロードする際にＳＭ初期化の一部として一度実行することができる。第２のメソッドについては、動的ＢＩＶの際にオンデマンドで必要とされるものをロードすることにより実行できる。これは、ＢＩＶがクラスについて必要とされる場合にできる。ＢＩＶデータファイルは、クラスレベルで組織化することができる。特定のクラスでは、ＢＩＶデータがこのクラスについてのみロードされかつ復号化される。この第２のメソッドは、ユーザに対してより自由度を与えて、クラスバイトコードが変化する場合にはＢＩＶデータに必要な小さな変化を与えることができる。

【０１０３】

バイトコード完全性検証動的セキュリティハンドラはまた動的ＢＩＶ１２１０を行う。上記のとおり、クラスまたはメソッドの動的ＢＩＶを、外部ＢＩＶＡＰＩコールまたは被保護Ｊａｖａアプリケーションから予め決まった内部ＢＩＶトリガを含むセキュリティモジュールへの他の機能コールにより起動される。動的ＢＩＶは、少なくとも以下のキーアクションを含む、すなわち、最新のバイトコードの入手、動的セキュアハッシュ値の計算およびタンパリング耐性ゲートキーパ（ＴＲＧＫ）１２１６を設けることである。

【０１０４】

最新のバイトコードの入手はアップコールによって行われる。セキュリティモジュール内のクラスまたはメソッドのためのセキュアな動的ハッシュ値を安全に計算するため、クラスまたはメソッドの最新のバイトコードを、ＪＮＩを介するＪＶＭ４１０へのアップコールにより得る必要がある。ＪＶＭ４１０へロードされるクラスまたはメソッドを実行しながら、同じバイトコード自体をバイナリに翻訳またはコンパイルする必要がある。バイトコードに対してタンパリング攻撃がない場合には、バイトコードは静的にセキュアなハッシュ値が計算された同じバイトコードのはずである。

【０１０５】

動的にセキュアなハッシュ値の計算アクションには、典型的で知られたハッシングの計算を含み、結果として得られる値は保護された形式であり、保護された形式で使用される。

【０１０６】

ＴＲＧＫ１２１６の提供には、２つの入力が含まれる。ＴＲＧＫ１２１６は、特定のクラスまたはメソッドについての静的および動的セキュアハッシュ値（ＳＳＨＶ１２１２、ＤＳＨＶ１２１４）の両方を使用し、クラスまたはメソッドのバイトコードの完全性が傷つけられているかどうかを検証する。バイトコードにタンパリングが起こっていれば、そのＤＳＨＶ１２１４は、そのＳＳＨＶ１２１２とは同じになり得ない。ＴＲＧＫ１２１６は、バイトコードに対するタンパリングを検出する。ＢＩＶ検証が通れば、ＴＲＧＫ１２１６は、成功のコールバック機能をトリガするかまたは元のＢＩＶトリガへ戻るか、そうでない場合はＴＧＲＫ１２１６は、ユーザの軽減アクションとして失敗コールバック機能をトリガすることになる。

【０１０７】

バイトコード完全性動的セキュリティハンドラ６１６は、また、ＢＩＶ終了という形の終了ステップを含む。セキュリティモジュールの一部としてのＢＩＶ終了により、メモリ空間およびＢＩＶ動的セキュアハンドラにより使用される他の情報のクリーンアップが行われる。

【０１０８】

図１３は、セキュアローディングバイトコード（ＳＬＢ、Ｓｅｃｕｒｅ　Ｌｏａｄｉｎｇ　Ｂｙｔｅｃｏｄｅ）静的セキュリティハンドラ５２８を示す。ＳＬＢ静的セキュリテ

ィハンドラ５２８が、元のＪａｖａアプリケーションバイトコード１３０１の内部表現をＷＢ暗号化１３０２と復号化キーデータ１３０４ならびに保護マーキング情報１３０６とともに受ける。

【０１０９】

ＳＬＢセキュリティハンドラ５２８の重要な出力がアプリケーションスタブ１３０８である。アプリケーションスタブ１３０８は、実行時セキュリティモジュールを介するロードのプロセスを起動するブートストラッピングクラスを含む。アプリケーションスタブ１３０８は、ＪＶＭ４１０によりロードされる。アプリケーションスタブ１３０８は、独立してまたは他のＪａｖａアプリケーションにより起動されるアプリケーションを有効にするのに必要な各パブリックＡＰＩを含む。アプリケーションスタブ１３０８は、セキュリティモジュールに対するダウンコール機能を起動するメソッドを含み、これは、次にＪａｖａアプリケーションをＪＶＭに復号化してロードし、実行する。

【０１１０】

アプリケーションスタブを準備するため、ＳＬＢ静的セキュリティハンドラ５２８は、アプリケーションバイトコード作業フレーム１３１０および暗号化されたアプリケーションバイトコード作業フレーム１３１２を含む。アプリケーションバイトコード作業フレーム１３１０は、もとのアプリケーションバイトコード１３０１とは異なる。一般に、アプリケーションバイトコード作業フレーム１３１０内のクラスは、保護を必要とせず、元のものと同じになる。クラスが安全にロードする必要がある場合、クラススタブは、元のクラスバイトコードを置き換え、それによりクラスバイトコードはもとのバイトコードとなる。暗号化されたフレーム１３１２は、構築時、静的セキュリティハンドラ５２８を介してアプリケーション暗号化キーデータ１３０２を使用してアプリケーションバイトコード作業フレーム１３１０を暗号化することにより得られ、かつ実行時、動的セキュリティハンドラ６２０を介してＷＢ復号化キーデータ１３０４を使用して復号化される。

【０１１１】

アプリケーションスタブ１３０８に加えて、アプリケーションペイロード１３１４が生成される。アプリケーションペイロード１３１４は、暗号化されたアプリケーション作業フレーム１３１２およびアプリケーションＷＢ復号化キーデータ１３１６を含む。被保護アプリケーションペイロードにおけるアプリケーションＷＢ復号化キーデータ１３１６は、ＷＢ静的セキュリティハンドラ５０８により生成されるキーデータであり、ＷＢ復号化キーデータ１３０２の一部としてＳＬＢ静的セキュリティハンドラ５２８に伝達される。実行時、暗号化されたアプリケーションバイトコード作業フレーム１３１２を復号化するために使用される。

【０１１２】

図１３に示す通り、基礎となるコードは、クラスバイトコード１３１８、クラススタブ１３２０または暗号化されたクラスバイトコード１３２２として形成され得る。クラスバイトコード１３１８は、元のバイトコードである。クラススタブ１３２０は、必要な場合暗号化されたクラスバイトコード１３２２をロードする実行時のセキュリティモジュールを介して信頼できるクラスローディングプロセスを起動するブートストラップ法を含む。パッケージの際に、クラスバイトコード１３１８が分析される。マークされたメソッドが、セキュリティモジュールにダウンコール法を起動するメソッドにより置換され、その場合セキュリティモジュールは、パッケージの際に指定されたセキュリティハンドラメソッドを介して元のバイトコード機能性を起動する。暗号化されたクラスバイトコード１３２２は、構築時、静的セキュリティハンドラ５２８によりクラスＷＢ暗号化キーデータ１３０２を使用してクラスバイトコード１３１８を暗号化することにより得られ、実行時、動的セキュリティハンドラ６２０を介してクラスＷＢ復号化キーデータを使用することにより復号化される。

【０１１３】

暗号化クラスバイトコードフレーム１３２４は、ＳＬＢ静的セキュリティハンドラ５２８により生成される。１以上のクラスについて暗号化クラスバイトコードおよびクラスＷ

Ｂ復号化キーを含む。ユーザは、暗号化されたクラスバイトコード１３２２内にフレームが含むことができるクラスの数を制御する選択肢を有する。ユーザは、これらを実行時ともにまたは別々にロードする選択肢を有する。クラスＷＢ復号化キーデータは、ＷＢ静的セキュリティハンドラ５０８により生成され、かつＷＢ復号化キーデータ１３０４の一部としてＳＬＢ静的セキュリティハンドラ５２８へ伝達される。実行時、クラスＷＢ復号化キーデータ１３０４を使用して、暗号化されたクラスバイトコード１３２２を復号化する。ユーザは、クラスＷＢ暗号化および復号化キーは１つまたは複数のいずれを生成するかという選択肢を有する。

【０１１４】

図１４に、ＳＬＢ動的セキュリティハンドラ６２０の作業フローを示す。ＳＬＢ動的セキュリティハンドラ６２０は、バイトコード実行の際にＪＮＩによりＪＶＭ４１０と接続されるセキュリティモジュールの機能要素である。ＳＬＢ動的セキュリティハンドラ６２０は、Ｊａｖａアプリケーションバイトコードの ＪＶＭ４１０における作業空間内へのロードを管理制御する。この能力により元のＪａｖａアプリケーションバイトコードが保護され、同様に保護された形式で配布され、ＪＶＭ４１０へロードされる前のアプリケーションバイトコードに対するいかなる静的攻撃をも確実に阻止することができる。ＳＬＢＤ-ハンドラ６２０は、セキュアなアプリケーションのロードおよびセキュアなクラスのロードを含む２つの主要機能要素を含む。

【０１１５】

セキュアなアプリケーションのロードには、クラスパスに存在しかつ通常ＪＶＭ４１０によりロードされる被保護アプリケーションスタブ１４０４を含む。ロードの後、メインブートストラップメソッドを実行し、その後アプリケーションブートストラップメソッド１４０３を、ＪＮＩＳＭブリッジ４１８経由でダウンコールＡＰＩを介して起動する。これにより、ＳＬＢ動的セキュリティハンドラ６２０の次のアプリケーションロードアクションがトリガされる。まず、被保護アプリケーションペイロード１４０８が保護データフォルダからロードされる。これは、暗号化されたアプリケーショバイトコード作業フレーム１４１０をペイロード１４０８からメモリバッファ内へロードすることと、次に、アプリケーションＷＢ復号化キーデータ１３０４を使用することによりジャストインメモリで、暗号化されたアプリケーションバイトコード作業フレーム１４１０を復号化することとを含む。次に、復号化されたアプリケーションバイトコード作業フレーム１４１２をウォークスルーして、特殊なＳＭクラスローダ１４１４を用いることによりアプリケーション作業空間へ作業フレームから各クラスバイトコードおよびクラススタブをロードする。ＳＭクラスローダ１４１４は、セキュリティモジュールを使用して暗号化されたバイトコードをロードし、このバイトコードを復号化しＪＶＭ４１０へロードする。追加のセキュリティチェックを組み込んで、ＳＭクラスローダ１４１４にＢＩＶ保護を加えかつロードおよび実行時のクラスローダのヒエラルキーおよび完全性についてチェックする。最後に、実行は、作業空間内のメインアプリケーションクラスのメインメソッドへ渡される。

【０１１６】

セキュアなクラスロードには、図１５に示すクラスブーストラップメソッドのトリガを含む。一般には、実行の際に、保護されるアプリケーションを実行する前に装置上に暗号化されたクラスバイトコードフレームをプリインストールまたはダウンロードすることができる。これは、アプリケーションの機能的性質に依存する。クラススタブを有するクラスは、被保護アプリケーションの実行の際に必要な場合、クラスブートストラップメソッドがトリガされ、暗号化されたクラスバイトコードフレームから必要なクラスをロードする次のステップ１５００がＪＮＩＳＭブリッジを介して実行される。まず、対応する暗号化されたクラスバイトコードフレームをメモリバッファにロードする。次に、フレームのジャストインメモリに含まれる暗号化されたクラスの各々を、特定のクラスＷＢ復号化キーデータを用いることにより復号化する。復号化したクラスバイトコードを、ＳＭクラスローダを用いてアプリケーション作業空間内へロードする。その後、アプリケーションの実行は、作業空間内で継続する。

【０１１７】
なお、すべてのコードをまずロードする必要があるネィティブアプリケーションを実行する場合と異なり、ＪＶＭが進行中に新たなクラスのロードを許可する。こうして、必要な場合にのみクラスをロードすることにより動的にアプリケーションを拡張する。その上、Ｊａｖａのこの特徴により、コードリフティング攻撃に対抗してＳＬＢセキュアクラスローディングを使用する良い機会が提供される。まず、被保護クラスがＳＬＢで安全にロードされ実行された後、本発明が、そのクラススタブに復元することによってクラスを保護状態に維持するオプションを提供できる。このように、まさに実行時のみに、クラスのオリジナルバイトコードがＪＶＭ画像内で入手可能な一方、それ以外の時には保護された形式のままである。

【０１１８】
ダイナミックバイトコードデクリプション（ＤＢＤ、Ｄｙｎａｍｉｃ　Ｂｙｔｅｃｏｄｅ　Ｄｅｃｒｙｐｔｉｏｎ）には、暗号化されたメソッドが、実行中のＪａｖａプログラムにより起動される場合にのみ被保護メソッドのバイトコードの復号化を含む。これにより、アプリケーションの暗号化されていないバイトコードのすべてが、一度にメモリに存在しないことが確実になる。

【０１１９】
図１６は、ＤＢＤ静的セキュリティハンドラ５３０の構築時作業フローを示す。構築時、各保護されていないクラスバイトコードファイル１６０２が、内部バッファへロードされ、新しいクラスバイトコード作業フレームが、保護マーキング情報１３０６を使用してＤＢＤにより保護されるクラスについて構築され、マークされたメソッドが、実行時にＤＢＤ動的セキュリティハンドラ５３０の起動をトリガするダウンコールメソッドを起動するメソッドスタブ１６０４と置き換えられる。保護対象の各Ｊａｖａメソッドについて、そのバイトコードはメソッドのＷＢ暗号化キーを使用して保護されたバイトコードデータの一部として配布するためのＷＢ復号化キーデータ１６０８と共にパッケージされた被暗号化メソッドを被暗号化メソッドバイトコードフレーム１６０６に記憶することにより暗号化される。オリジナルバイトコードクラスは被保護クラスバイトコード作業フレーム１６１０により置き換えられて配布される。

【０１２０】
図１８は、ＤＢＤ動的セキュリティハンドラ６２２の実行時間作業フローである。ＪＶＭ上で、被保護Ｊａｖａアプリケーションを実行中に暗号化されたＤＢＤＪａｖａメソッドが起動されると、メソッドスタブがまず実行されて、その後ダウンコールが行われ、メソッドブートストラッピング１８０２が、ＤＢＤ動的セキュリティハンドラ６２２内で起動される。これは、ＷＢメソッド復号化キーデータを使用することにより被暗号化メソッドを被暗号化メソッドバイトコードフレーム１８０４から識別しかつ復号化し、その真のバイトコードをＪＶＭへ復元する例を含む。ＪＶＭへクラスバイトコードを復元するための実現例には、ＪＶＭネームスペース内での名称の混乱を避けるためクラスを再命名して、ＪＶＭへクラスのコピーを再構成する。図１７にこの例を示す。図中、部分的に復号化されたクラスが新たなクラス名とともにＪＶＭにロードされる。

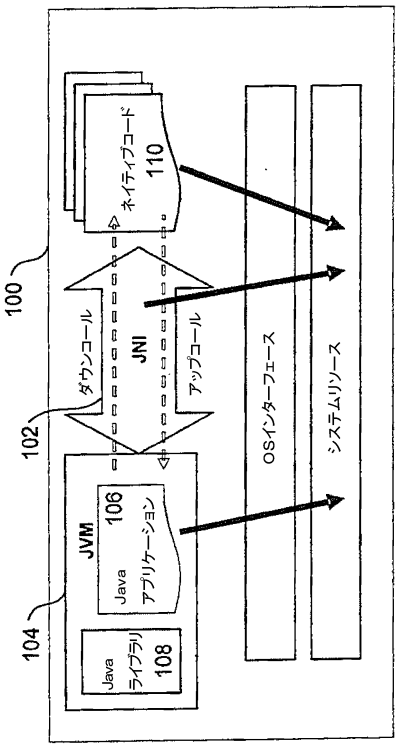
【０１２１】
図１８において、必要であれば、オリジナルのバイトコードがＪＶＭに復元されると、ＤＢＤ動的セキュリティハンドラ６２２が、実バイコードインスタンスにクラスの状態をコピーすることができ、このオプションは、構築時に決定される。ＤＢＤ動的セキュリティハンドラ６２２は、そこでＪＶＭ４１０における非暗号化メソッドを起動する。メソッドの起動が終了すると、セキュリティハンドラ６２２は、非暗号化クラスインスタンスからの実状態を暗号化インスタンスに復元し、制御は起点のダウンコールメソッドへ戻る。図１７は、非暗号化メソッドを呼び出す前のサンプルのメソッド起動および状態コピーオペレーション１７００を示す。非暗号化メソッドが実行を終えると、その状態をＤＢＤ動的セキュリティハンドラ６２２により被保護メソッドスタブでクラスインスタンスへ再びコピーする。制御は、被保護メソッドへ戻り、一方セキュリティハンドラは非暗号化クラ

スおよびインスタンスをＪＶＭ４１０から削除する。

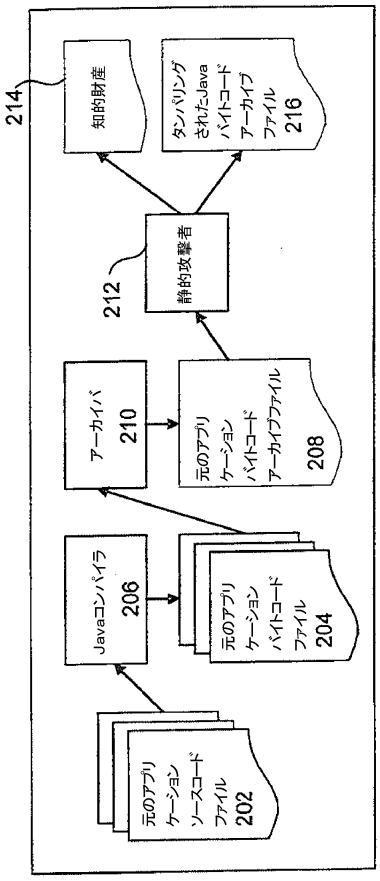
【０１２２】

本発明の上記の実施例は、例示目的のみのものである。当業者においては、添付の特許請求の範囲によってのみ規定される発明の範囲から逸脱することなく、特定の実施例には変更、修正および変形が可能である。

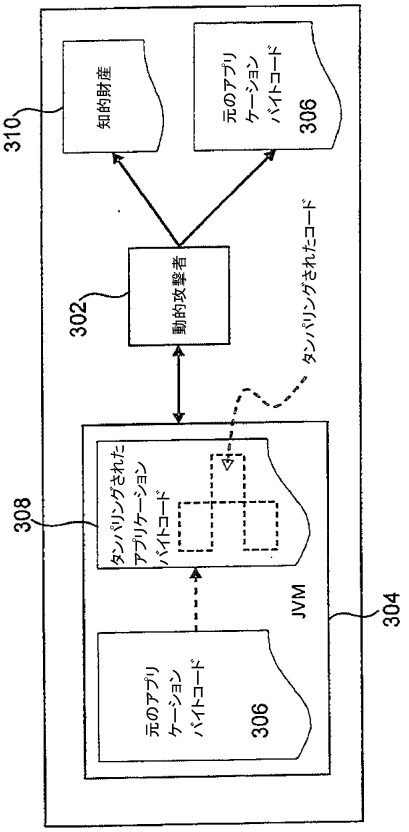
【図１】



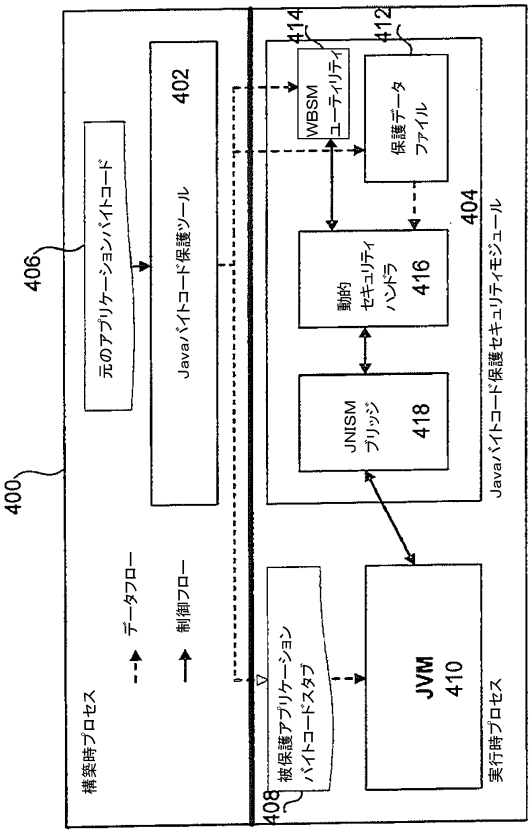
【図２】



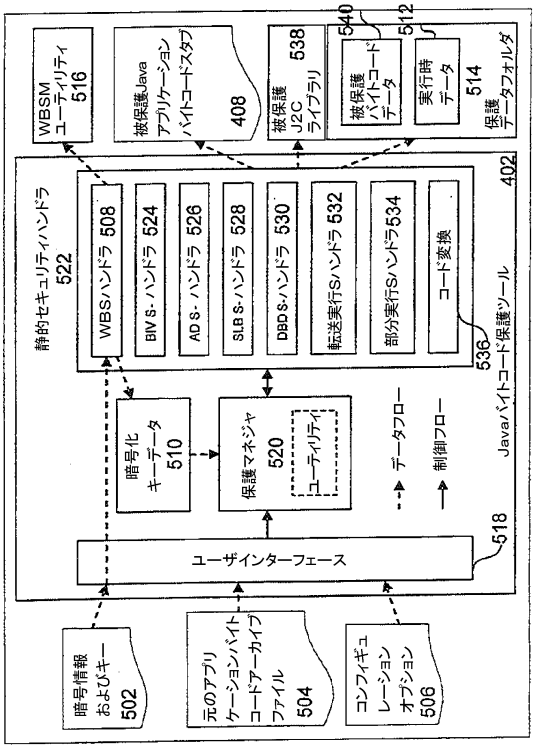
【図３】


310
知的財産
元のアプリケーションバイトコード
306
302
動的攻撃者
タンパリングされたコード
308
タンパリングされたアプリケーションバイトコード
元のアプリケーションバイトコード
306
JVM
304

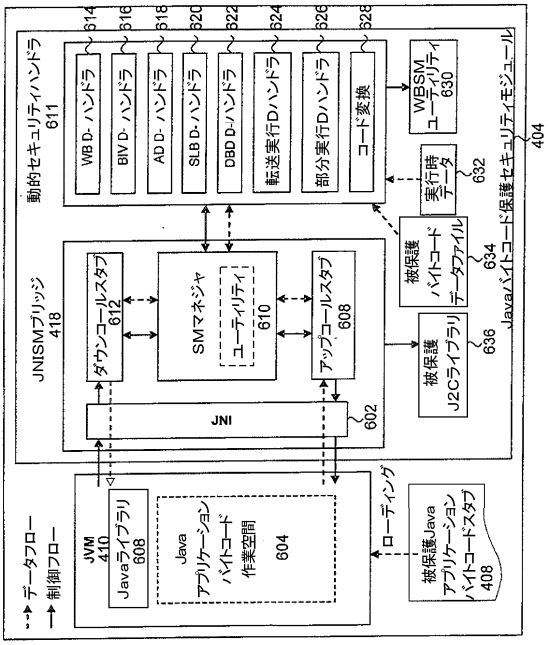
【図４】


400
406
元のアプリケーションバイトコード
Javaバイトコード保護ツール
402
414
WBSM ユーティリティ
412
保護データファイル
404
動的セキュリティハンドラ
416
JNISMブリッジ
418
Javaバイトコード保護セキュリティモジュール
構築時プロセス
データフロー
制御フロー
被保護アプリケーションバイトコードスタブ
408
JVM
410
実行時プロセス

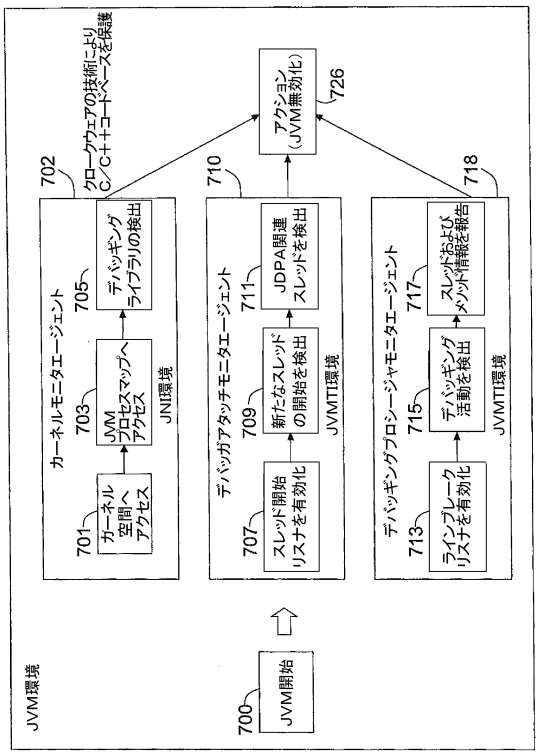
【図５】


WBSM ユーティリティ 516
被保護Java アプリケーションバイトコードスタブ 408
被保護 J2C ライブラリ 538
被保護バイトコードデータ 540
実行時データ 512
保護データフォルダ 514
静的セキュリティハンドラ 522
WBSハンドラ 508
BIV S-ハンドラ 524
AD S-ハンドラ 526
SLB S-ハンドラ 528
DBD S-ハンドラ 530
転送実行S-ハンドラ 532
部分実行S-ハンドラ534
コード変換
402
536
暗号化キーデータ 510
保護マネジャ 520
ユーティリティ
データフロー
制御フロー
Javaバイトコード保護ツール
518
ユーザインターフェース
暗号情報およびキー 502
元のアプリケーションバイトコードアーカイブファイル 504
コンフィギュレーションオプション 506

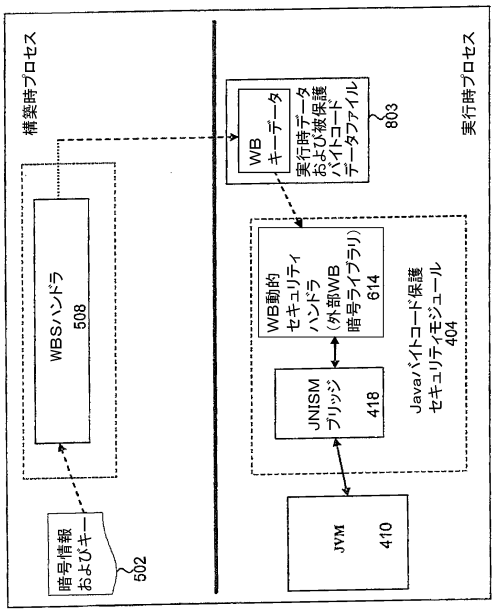
【図６】


データフロー
制御フロー
JVM 410
Javaライブラリ 608
Java アプリケーションバイトコード作業空間 604
ローディング
被保護Java アプリケーションバイトコードスタブ 408
JNISMブリッジ 418
JNI
602
ダウンコールスタブ 612
SMマネジャ
ユーティリティ 610
アップコールスタブ 608
被保護 J2Cライブラリ 636
動的セキュリティハンドラ 611
WB D-ハンドラ 614
BIV D-ハンドラ 616
AD D-ハンドラ 618
SLB D-ハンドラ 620
DBD D-ハンドラ 622
転送実行Dハンドラ 624
部分実行Dハンドラ 626
コード変換 628
WBSM ユーティリティ 630
実行時データ 632
被保護バイトコードデータファイル 634
Javaバイトコード保護セキュリティモジュール
404

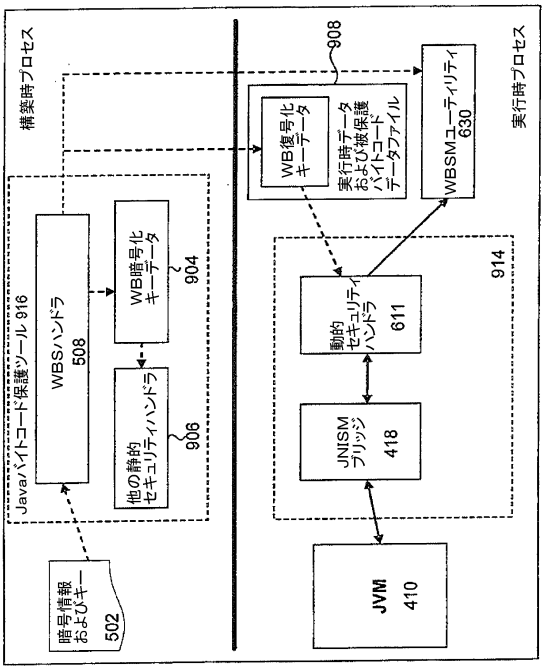
【図７】


JVM環境
700
JVM開始
カーネルモニタエージェント
702
701
カーネル空間へアクセス
703
JVMプロセスマップへアクセス
705
デバッギングライブラリの検出
JNI環境
デバッガアタッチモニタエージェント
710
707
スレッド開始リスナを有効化
709
新たなスレッドの開始を検出
711
JDPA関連スレッドを検出
JVMTI環境
デバッギングプロシージャモニタエージェント
715
713
ラインブレークリスナを有効化
717
デバッギング活動を検出
スレッドおよびメソッド情報を報告
718
アクション（JVM無効化）
726
クロークウェアの技術によりC/C++コードベースを保護

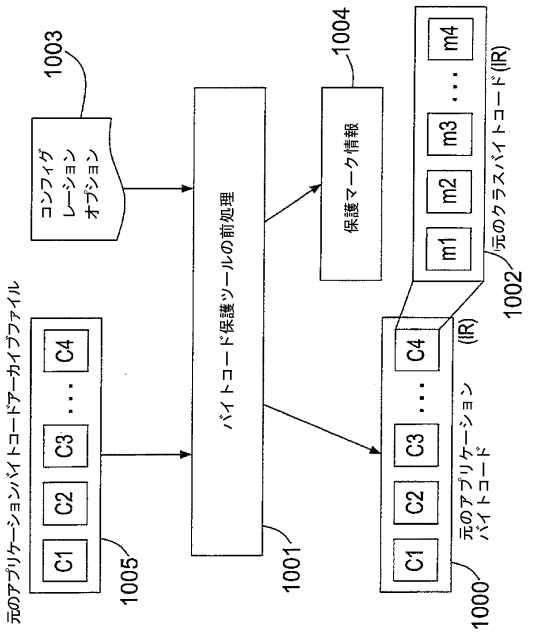
【図８】


構築時プロセス
実行時プロセス
WBSハンドラ
508
暗号情報およびキー
502
WB
キーデータ
実行時データおよび被保護バイトコードデータファイル
803
WB動的セキュリティハンドラ（外部WB暗号ライブラリ）
614
JNISMブリッジ
418
Javaバイトコード保護セキュリティモジュール
404
JVM
410

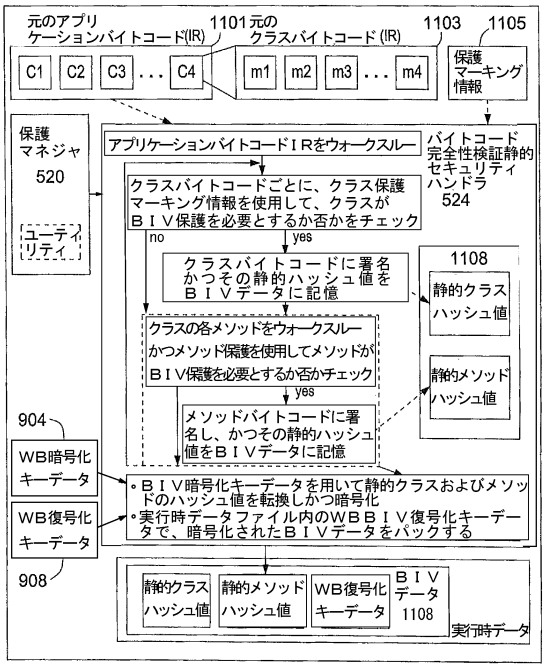
【図９】


構築時プロセス
実行時プロセス
Javaバイトコード保護ツール 916
WBSハンドラ
508
WB暗号化キーデータ
904
他の静的セキュリティハンドラ
906
暗号情報およびキー
502
908
WB復号化キーデータ
実行時データおよび被保護バイトコードデータファイル
WBSMユーティリティ
630
914
動的セキュリティハンドラ
611
JNISMブリッジ
418
JVM
410

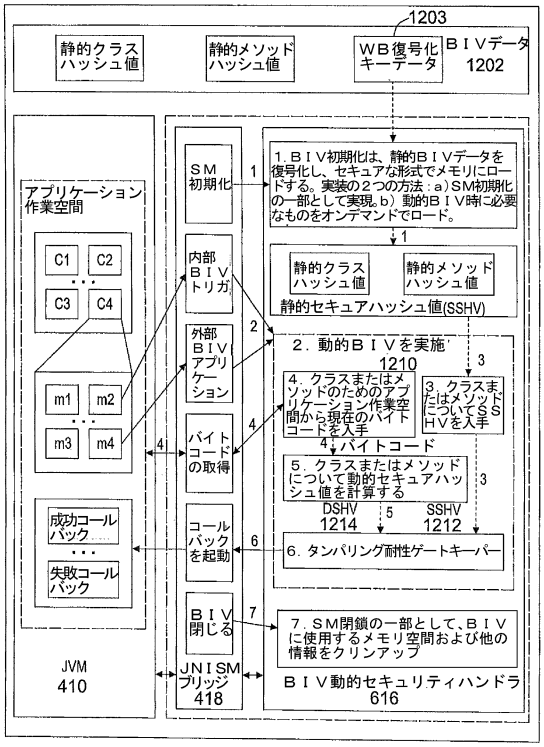
【図１０】


元のアプリケーションバイトコードアーカイブファイル
C1
C2
C3
C4
1005
コンフィグレーションオプション
1003
バイトコード保護ツールの前処理
1001
保護マーク情報
1004
m1
m2
m3
m4
元のクラスバイトコード(IR)
1002
元のアプリケーションバイトコード (IR)
1000

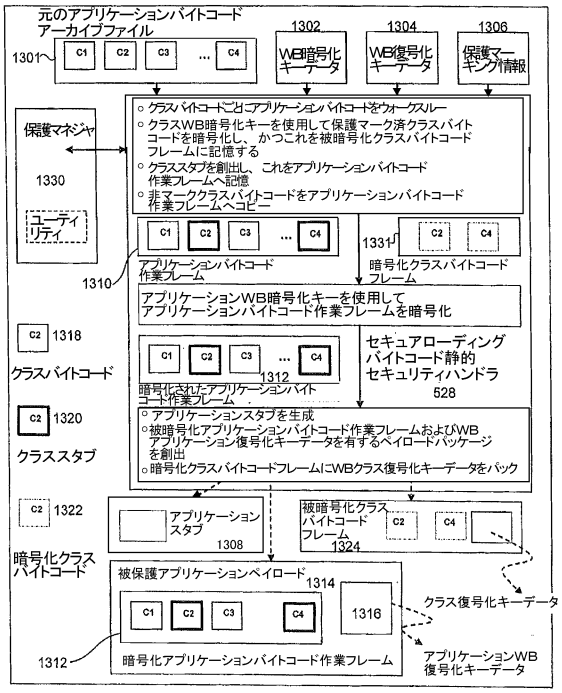
【図１１】



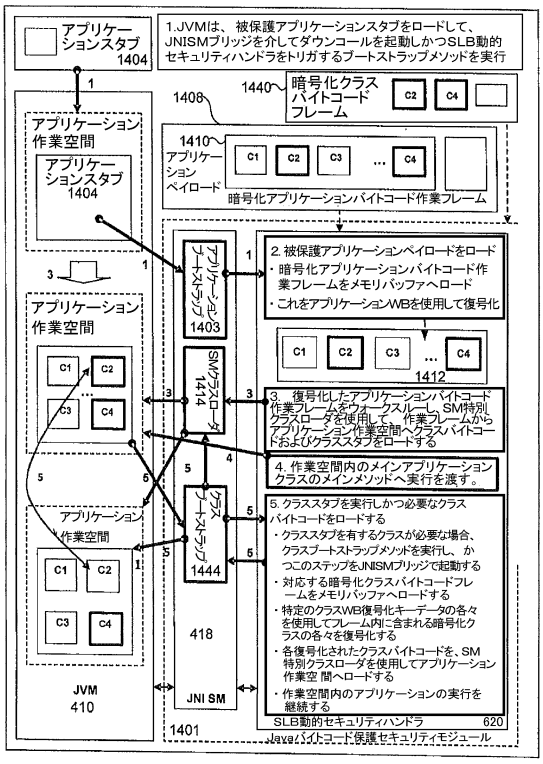
【図１２】



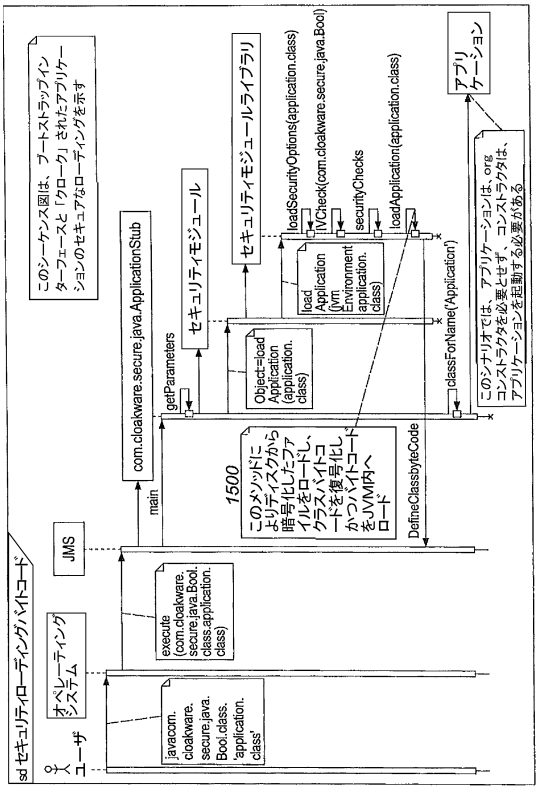
【図１３】



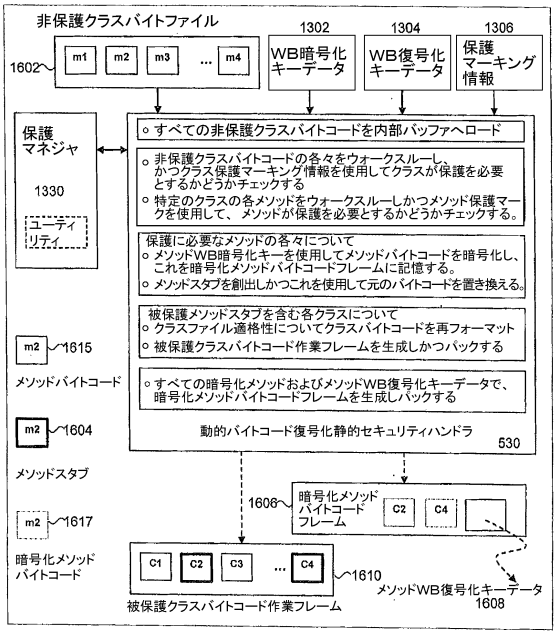
【図１４】



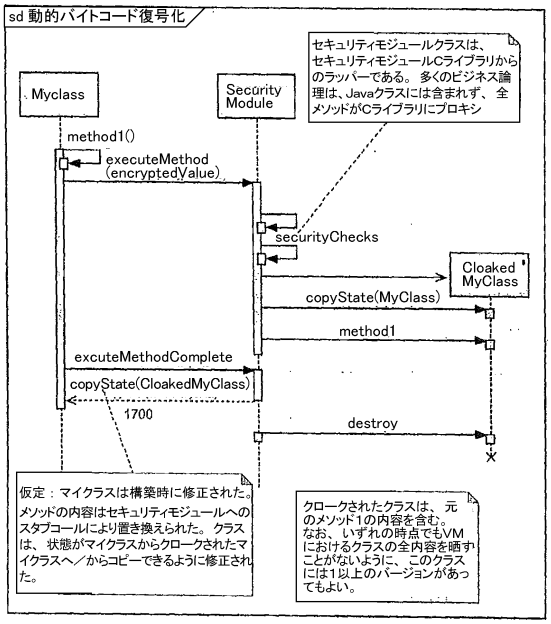
【図１５】



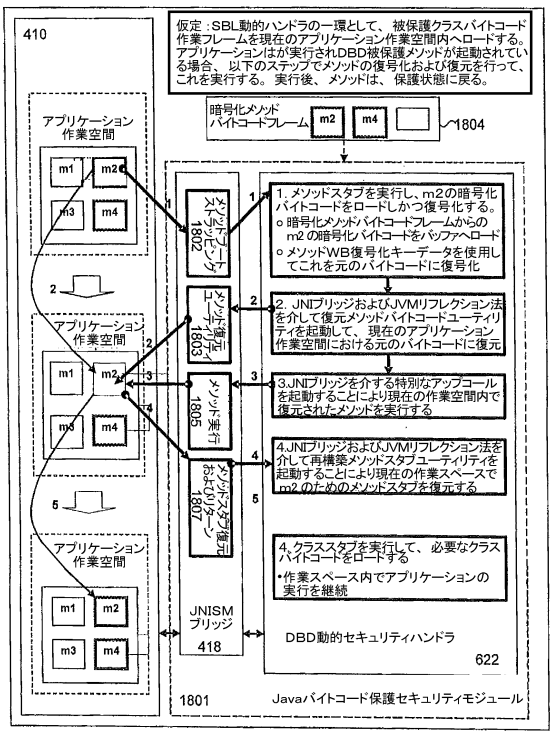
【図１６】



【図１７】



【図１８】



フロントページの続き

(74)代理人 100120662
弁理士 川上 桂子
(72)発明者 グ ユエン シャン
カナダ ケー2ティー 1ジー5, オンタリオ, カナタ, インスミル クレセント 39
(72)発明者 ガーネイ アダムス
カナダ ケー2エス 2エイチ6, オンタリオ, スティッツヴィレ, アップカントリー ドライブ 304
(72)発明者 ジャック ロン
カナダ ケー2ダブリュ 0エー5, オンタリオ, カナタ, ウィンダンス クレセント 179

審査官 木村 励

(56)参考文献 特開2009－258772（JP, A）
米国特許出願公開第2004／0039926（US, A1）
国際公開第2009／095838（WO, A1）
国際公開第2007／147495（WO, A1）
特開2005－293109（JP, A）

(58)調査した分野(Int.Cl., DB名)
G06F 21／00 － 21／88