(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3572016号
(P3572016)
(45)発行日　平成16年9月29日(2004.9.29)　(24)登録日　平成16年7月2日(2004.7.2)

(51)Int.Cl.[7]	F I		
G06F 9/45	G06F	9/44	320C
G06F 1/00	G06F	12/14	510D
G06F 12/14	G06F	9/06	660Z

請求項の数 1　(全 13 頁)

(21)出願番号	特願2000-508048(P2000-508048)
(86)(22)出願日	平成10年8月25日(1998.8.25)
(65)公表番号	特表2001-514411(P2001-514411A)
(43)公表日	平成13年9月11日(2001.9.11)
(86)国際出願番号	PCT/US1998/017553
(87)国際公開番号	WO1999/010795
(87)国際公開日	平成11年3月4日(1999.3.4)
審査請求日	平成12年2月28日(2000.2.28)
(31)優先権主張番号	08/919,844
(32)優先日	平成9年8月28日(1997.8.28)
(33)優先権主張国	米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　ボンド　バリー
アメリカ合衆国　ワシントン州　９８０５９　レントン　ノースイースト　トゥエンティファースト　ストリート　４９０２

最終頁に続く

(54)【発明の名称】信頼されないプログラムを実行するための方法

(57)【特許請求の範囲】
【請求項１】
メモリおよびインターフェースモジュールを有するパーソナルコンピュータプラットフォームで実行するために書かれた信頼されない第２のプログラムを、ＣＰＵを有するコンピュータが実行可能な第１のプログラムに変換して実行するための方法であって、
前記ＣＰＵにより前記第２のプログラムに関してメモリの所定境界領域を割り当て、
前記ＣＰＵにより前記第２のプログラムを前記メモリの境界領域内にロードし、
当該ロードされた第２のプログラム中の、前記インターフェースモジュールがリンク先となっている命令を前記ＣＰＵにより検出し、当該検出した命令のリンク先を、前記インタフェースモジュールから、該命令をブロックするための変換コードモジュールに前記ＣＰＵにより置換し、
当該置換が行われた第２のプログラムを第１のプログラムに変換し、
当該変換された第１のプログラムの中の前記メモリ境域以外の領域に対してリファレンスを行う命令を前記ＣＰＵにより検出し、当該検出した命令の実行時のリファレンスをブロックするためのチェックコードを前記第１のプログラムの中に前記ＣＰＵにより配置し、
前記チェックコードが配置された第１のプログラムを前記ＣＰＵにより実行する
ことを特徴とする方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】

本発明は、電子的データ処理に関し、特に、信頼できないコードを包含する実行可能プログラムからのシステム損傷を回避することに関する。

【０００２】

【従来の技術】

インターネットブラウザの進歩は、ＷＷＷ（ワールドワイドウェブ：Ｗｏｒｌｄ Ｗｉｄｅ Ｗｅｂ）の動的及びインタラクティブなページを作り出す。しかしながら、進歩はまた、ウェブページを単に見ることから発生する多くのコンピュータシステムセキュリティリスクを作り出す。インターネットブラウザは、プログラム又は、ウェブページに埋め込まれた他の実行可能なコードを自動的にダウンロードし、実行する。リモートコンピュータからプログラムをダウンロードし、実行する能力は、ホストコンピュータを種々のセキュリティリスクに曝す。例えばコンピュータシステム又は、コンピュータシステムのデータを修正する敵意のあるプログラムは、パスワード、銀行預金口座情報のようなユーザデータを盗み、ユーザにシステムリソースを利用できなくさせる。その結果、セキュリティの問題は、インターネットアプリケーションの開発において重要である。

【０００３】

ある従来技術のアプローチにより、Ｊａｖａアプレットとして知られる実行可能なコードの特定のフォームにセキュリティを設けた。実行可能なコードソースプログラムは、書き込まれ、プラットフォーム独立バイトコードに変換されダウンロードされる。プラットフォーム独立トークン化されたバイトコードは、実行可能コードがすることができる厳格な制限を配置する仮想マシーンで走る。従来技術のアプローチにおける実行可能なコードは、オペレーティングシステムへのアクセスが非常に制限されていた。従って、Ｊａｖａ言語はより強力になるので、オペレーティングシステムがすでに実行できる多くの関数を複製しなければならない。

【０００４】

ＡｃｔｉｖｅＸコントロールは、Ｊａｖａの制限された能力を回避する実行可能なコードのフォームである。ＡｃｔｉｖｅＸは、ＯＬＥ（Ｏｂｊｅｃｔ Ｌｉｎｋｉｎｇ ａｎｄ Ｅｍｂｅｄｄｉｎｇ）及びＣＯＭ（Ｃｏｍｐｏｎｅｎｔ Ｏｂｊｅｃｔ Ｍｏｄｅｌ）と呼ばれるマイクロソフト社の２つの技術の産物である。ＡｃｔｉｖｅＸは、それがインターネットを利用することができるという特徴をサポートする。例えば、ＡｃｔｉｖｅＸコントロールは、Ｗｅｂブラウザによって自動的にダウンロードされ、実行される。

【０００５】

ＡｃｔｉｖｅＸコントロールが、ネイティブコードで書かれるので、それらはオペレーティングシステムに十分にアクセスでき、コントロールが稼働するメモリを処理することができる。このアクセスは、コントロールが、スタンドアロンアプリケーションに対する拡張のようなきつく制御された環境で稼働するときに、強力である。しかしながら、ＡｃｔｉｖｅＸコントロールが、インターネットエクスプローラのようなウェブブラウザのようなアプリケーションによってインターネット上の知らない又は信用できないソースからダウンロードされるとき、オペレーティングシステムへの十分なアクセスは、深刻なセキュリティの問題を生ずる。ＡｃｔｉｖｅＸコントロールは、いかなるオペレーティングシステムのサービスにもアクセスするように設計される。敵意のあるＡｃｔｉｖｅＸコントロールは、ホストシステムのハードドライブのお情報を検索することができ、ウィルスを注入することができ、又は、ホストシステムを損傷させることができた。オペレーティングシステムに対するＡｃｔｉｖｅＸの無制限のアクセスによる問題は、無制限のアクセスが、セキュリティ違反に対するリスクにホストシステムを置くことである。

【０００６】

従って、ホストシステムのセキュリティを妥協することのない、ホストオペレーティングシステムのパワーにアクセスする能力を備えた実行可能なコードのフォームの必要性がある。

【０００７】

【発明が解決しようとする課題】

本発明は、ネイティブ、即ち直接実行可能なコードで書かれた信頼されたい実行可能なコードに関するセキュリティポリシーを実行する。実行可能なコードは、メモリの外側へのリファレンスが制限される予め割り当てられたメモリ範囲、即ちサンドボックス内にロードされる。実行中、実行可能なコードに追加されたチェック（「スニフコード（ｓｎｉｆｆ ｃｏｄｅ）」）は、これらの制限を強制する。信頼されないコードにおける在来のアプリケーションプログラムインターフェース（ＡＰＩ）コールは、ホストシステムのセキュリティの侵害を防止しながら、実行コードがホストオペレーティングシステムにアクセスすることができる変換コードモジュール（「サンク（ｔｈｕｎｋｓ）」）で置換される。コントロール又はアプレットにおける静的リンクは、コールによってサンクモジュールに置換される。実行中にＡＰＩコールが作られるとき、コントロールはサンクに移送し、ＡＰＩコールがオペレーティングシステムで実行されることが許容されるべきか否か判断する。

【０００８】

【課題を解決するための手段】

本発明は、メモリおよびインターフェースモジュールを有するパーソナルコンピュータプラットフォームで実行するために書かれた信頼されない第２のプログラムを、ＣＰＵを有するコンピュータが実行可能な第１のプログラムに変換して実行するための方法であって、前記ＣＰＵにより前記第２のプログラムに関してメモリの所定境界領域を割り当て、前記ＣＰＵにより前記第２のプログラムを前記メモリの境界領域内にロードし、当該ロードされた第２のプログラム中の、前記インターフェースモジュールがリンク先となっている命令を前記ＣＰＵにより検出し、当該検出した命令のリンク先を、前記インタフェースモジュールから、該命令をブロックするための変換コードモジュールに前記ＣＰＵにより置換し、当該置換が行われた第２のプログラムを第１のプログラムに変換し、当該変換された第１のプログラムの中の前記メモリ境域以外の領域に対してリファレンスを行う命令を前記ＣＰＵにより検出し、当該検出した命令の実行時のリファレンスをブロックするためのチェックコードを前記第１のプログラムの中に前記ＣＰＵにより配置し、前記チェックコードが配置された第１のプログラムを前記ＣＰＵにより実行することを特徴とする。

【発明の実施の形態】

以下の実施の形態の詳細な説明において、添付の図面を参照するが、それらは、本発明を実施するための特定の実施形態の例示として示したものである。これらの実施形態は、当業者が本発明を実施するのに十分に詳細に記載されており、他の実施形態が利用可能であり、本発明の精神及び範囲を逸脱することなく論理的及び電気的な変更をすることができることを理解すべきである。それゆえ、以下の詳細な説明を限定的な意味にとってはならず、本発明の範囲は特許請求の範囲のみによって定義される。複数の図で表される同一のコンポーネントは同じ参照番号によって識別される。

【０００９】

図１及び以下の議論は、本発明を実行することができる適当な計算環境の一般的な説明を短く提供するものである。望まないけれども、本発明は、パーソナルコンピュータによって実行されるプログラムモジュールのようなコンピュータ実行可能な命令の一般的なコンテキストで記載される。一般的に、プログラムモジュールは、特定のタスクを実行し、又は、特定の抽象データ型を実行する、ルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを包含する。更に、本発明が、ハンドヘルドデバイス、マルチプロセッサシステム、マイクロプロセッサベース又はプログラム可能なカスタマエレクトロニック、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、などを包含する他のコンピュータシステムで実行されうることは、当業者には明らかであろう。本発明はまた、通信ネットワークを介してリンクされたリモート処理デバイスによってタスクが実行される分散計算環境でも実行される。分散計算環境では、プログラムモジュールは、ローカルとリモートの両方のメモリ記憶装置に配置される。

【００１０】

図１は、本発明が実施される適当な計算環境の簡単な一般的な説明を提供する。本発明は

、以下において、他の環境でも可能であるが、パーソナルコンピュータ（ＰＣ）によって実行されるプログラムモジュールのようなコンピュータ実行可能な命令の一般的なコンテキストとして記載する。プログラムモジュールは、特定のタスクを実行し、特定の抽象データ型を実行するルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを包含する。本発明が、ハンドヘルドデバイス、マルチプロセッサシステム、マイクロプロセッサベース又はプログラム可能なカスタマエレクトロニック、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、などを包含する他のコンピュータシステムで実行されうることは、当業者には明らかであろう。本発明はまた、通信ネットワークを介してリンクされたリモート処理デバイスによってタスクが実行される分散計算環境でも実行される。分散計算環境では、プログラムモジュールは、ローカルとリモートの両方のメモリ記憶装置に配置される。

【００１１】

図１は、本発明を実行するためのシステムの例を示す。それは従来のパーソナルコンピュータ２０のフォームにおいて、汎用のコンピュータデバイスを採用し、該コンピュータは、演算ユニット２１と、システムメモリ２２と、システムメモリ及び他のシステムコンポーネントを演算ユニット２１に接続するシステムバス２３とを含む。システムバス２３は、メモリバス又はメモリコントローラ、周辺バス、及び、ローカルバスを包含する種々のタイプのものであってよく、多数のバス構造を使用するものであってよい。システムメモリ２２は、ＲＯＭ２４とＲＡＭ２５を包含する。ＲＯＭ２４にストアされた基本入力／出力システム（ＢＩＯＳ）は、パーソナルコンピュータ２０のコンポーネントの間に情報を転送する基本ルーチンを包含する。ＢＩＯＳ２４はまた、システムのスタートアップルーチンを包含する。パーソナルコンピュータは更に、ハードディスク（図示せず）から読み出し、該ディスクに書き込むハードディスクドライブ２７と、リムーバブル磁気ディスク２９から読み出し、該ディスク２９に書き込む磁気ディスクドライブ２８と、ＣＤ－ＲＯＭ又は他の光学媒体のようなリムーバブル光ディスク３１から読み出し、該ディスク３１に書き込む光ディスクドライブ３０とを包含する。ハードディスクドライブ２７、磁気ディスクドライブ２８、及び、光ディスクドライブ３０は、ハードディスクドライブインターフェース３２と磁気ディスクドライブインターフェース３３と、光ディスクドライブインターフェース３４のそれぞれによってシステムバス２３に接続される。ドライブ及びそれらの関係するコンピュータ読み取り可能媒体は、コンピュータ読み取り可能命令、データ構造、プログラムモジュール、及び、パーソナルコンピュータ２０に関する他のデータの不揮発的なストレージを提供する。ここで記載した例示的な環境は、ハードディスク、リムーバブル磁気ディスク２９及びリムーバブル光ディスク３１を採用するけれども、コンピュータによってアクセス可能なデータをストアすることができる他のタイプのコンピュータ読み取り可能媒体をまた具体的な操作環境に使用することもできることは当業者にとって明らかであろう。かかる媒体は、磁気カセット、フラッシュメモリカード、ディジタル汎用ディスク、ベルヌーイカートリッジ、ＲＡＭ、ＲＯＭ及び同様なものを包含する。

【００１２】

プログラムモジュールは、ハードディスク、磁気ディスク２９，光ディスク３１、ＲＯＭ２４、及びＲＡＭ２５にストアされうる。プログラムモジュールは、オペレーティングシステム３５と、１又はそれ以上のアプリケーションプログラム３６と、他のプログラムモジュール３７と、プログラムデータ３８とを包含しうる。ユーザは、ユーザは、キーボード４０及びポインティングデバイス４２のような入力デバイスを介してコマンド及び情報をパーソナルコンピュータに入力する。他の入力デバイス（図示せず）は、マイクロホン、ジョイスティック、ゲームパッド、衛生アンテナ（サテライト・ディッシュ）、スキャナ等を包含する。これら及び他の入力デバイスはしばしば、システムバス２３に接続されたシリアルポートインターフェース４６を介して演算ユニット２１に接続されるが、それらは、パラレルポート、ゲームポート、又はユニバーサルシリアルバス（ＵＳＢ）のような図１に示されていない他のインターフェースを介して接続されうる。モニタ４７又は他

の表示デバイスまたは、ビデオアダプタ４８のようなインターフェースを介してシステムバス２３に接続されうる。モニタに加え、パーソナルコンピュータは典型的には、スピーカ及びプリンタのような楽しゅう偏出力デバイス（図示せず）を含む。

【００１３】

パーソナルコンピュータ２０は、リモートコンピュータ４９のような１又はそれ以上のリモートコンピュータに対して論理的な接続を使用してネットワークされた環境で使用されうる。リモートコンピュータ４９は、他のパーソナルコンピュータ、サーバ、ルータ、ネットワークコンピュータ、ピア・デバイス、又は他の一般的なネットワークノードであってよい。それは典型的には、パーソナルコンピュータ２０と接続する上述の多くの又は全てのコンポーネントを包含するが、記憶装置５０だけを図１に例示した。図１に示した論理的な接続は、ローカルエリアネットワーク（ＬＡＮ）５１及びワイドエリアネットワーク（ＷＡＮ）５２を含む。かかるネットワーク環境は、オフィス、企業の広汎なネットワーク、イントラネット及びインターネットでありふれている。

【００１４】

ＬＡＮネットワーク環境に配置されたとき、ＰＣ２０は、ネットワークインターフェイス又はアダプタ５３を介してローカルネットワーク５１に接続する。インターネットのようなＷＡＮネットワーク環境で使用されるとき、ＰＣ２０は典型的にはネットワーク５２にわたって通信を確立するためのモデム５４又は他の手段を包含する。モデム５４は、ＰＣ２０の内部又は外部にあってよく、シリアルポートインターフェース４６を介してシステムバス２３に接続する。ネットワーク化された環境では、２０内に存在するように示されたプログラムモジュール又はその一部は、リモート記憶装置５０にストアされうる。もちろん、ネットワーク接続は例示的に示されたものであり、コンピュータ間の通信リンクを確立する他の手段で置換することができる。

【００１５】

本発明では、アプリケーションプログラム３６としてパーソナルコンピュータ２０で稼働する在来のウェブブラウザが、リモートコンピュータ４９からアプレットを自動的にダウンロードする。「アプレット」は短いプログラムであり、通常は単一の関数で実行され、他のアプリケーション内で実行されるように設計されている。アプレットは、それらが必要なときにリモートコンピュータからしばしばダウンロードされ、それらがプライマリアプリケーションによって実行された後、ローカルコンピュータからときどき消去されうる。

【００１６】

図２は、本発明における、アプレットを稼働するためのファシリティを含む殆どが在来の実行環境を示す。用語「アプレット」は、従来技術において正確に定義されていない。この用語は一般的には、単一の関数又は制限された範囲の関数を実行するための小さなプログラムと呼ばれるが、用語は本来は、プログラムのサイズ又はその関数の範囲を制限されない。アプレットは、特定の目的でＷＷＷページのようなオンラインソースからしばしばダウンロードされ、実際には、アプレットはダウンロードされるとすぐに実行され、ついで実行後削除される。以下に記載する好ましい実施形態では、用語「コントロール」又は「ＡｃｔｉｖｅＸコントロール」は、アプレットと同意語と考えて良い。ある場合では、発明それ自身が小さなプログラム、ダウンロードされたプログラム、又は、他のいかなるプログラムの特定のフォームで使用することを制限しない。本発明は、「信頼される」ことがないいかなるプログラムについても有用である、即ち、該プログラムとは、システムリソースに十分にアクセスしたならば、システムを損傷させるかも知れない、不確実な出所又は効果のプログラムである。

【００１７】

Ｗｉｎｄｏｗｓ９５のようなオペレーティングシステム３５は、通常のアプリケーションプログラム３６をメモリ内にロードするためのローダモジュール３５１を採用する。プログラム３６は、ライン３６１によって表されるような演算ユニット２１に命令を直接送信することによって、オペレーティングシステム３５の制御下で実行する。プログラム３６

は、アプリケーションプログラムインターフェース（ＡＰＩ）コード３５２－３５４のブロックを呼び出すことによって標準のＡＰＩファンクションを実行する。各ＡＰＩは、図１のディスプレィ４７にダイアログボックスを表示するように、特定のレベルのファンクションを実行するためのプロセッサ２１によって直接実行可能な命令を包含する。ＯＳ３５は、一般的に、数千の独立したＡＰＩを含んでおり、数ダースのダイナミックリンクライブラリ（ＤＬＬ）として通常パッケージングされており、Ｍｉｃｒｏｓｏｆｔ ＷｉｎｄｏｗｓＮＴオペレーティングシステムでは、これらのＤＬＬは、集合名詞的に「Ｗｉｎ３２」として知られている。

【００１８】

エミュレータプログラムによって、ある演算ユニット２１用に書かれたアプリケーションプログラムが、異なる命令セットを有する別の演算ユニットで実行される。ここで採用する特定のＷｘ８６ＶＭエミュレータ３９は、インテル「ｘ８６」プロセッサ（８０３８６，８０４８６，ペンティアムなど）用に書かれたプログラムをＤｉｇｉｔａｌ Ｅｑｕｉｐｍｅｎｔ Ｃｏｒｐ．のＡｌｐｈａ及びＩＢＭのＰｏｗｅｒＰＣのようなプロセッサで実行するためにオリジナルに開発されたものである。それについては、出願中であるシリアル番号０８／９１２，４５４及び０８／９０４，０５７により詳細に記載してある。本目的に関して、Ｗｘ８６ＶＭと呼ばれるいくぶん修正されたバージョンは、殆どの修正されていない命令をｘ８８プロセッサ２１に通すが、記載するようなその他をブロックし、変換する。Ｗｘ８６ＶＭは、「サンクコード」（又は単に「サンク」）３９１－３９３と呼ばれる変換モジュールによってＡＰＩを実行する際にＷｘ８６をまねるが、ここでのサンクコードの目的は、セキュリティを提供することであり、異なるプラットフォーム用に書かれたＡＰＩコードを実行するために、あるプラットフォームからＡＰＩコールをすることができるというそれらの本来の目的ではない。

【００１９】

３６２のようなアプレットが実行されるとき、インターネットウェブブラウザのようなホストプログラム３６は、エミュレータ３９を呼び出す。エミュレータは、アプレットコードを所定のメモリ領域にロードするため、及びそれの使用のために別の所定のメモリ領域を割り当てるために、それ自身のローダモジュール３９６を採用する。これらの領域は、そのアプレットに関する「サンドボックス（ｓａｎｄｂｏｘ）」と呼ばれる。アプレットの実行中、エミュレータ３９は、アプレットのコードを、サンドボックスの外側に存在するコンパイルされたキャッシュにコンパイルする。コンパイルプロセス中、エミュレータはまた、メモリスニフ（ｓｎｉｆｆ）コード３９４をキャッシュ内に挿入する。

【００２０】

アプレット３６２は、それが書かれた同じプロセッサプラットフォーム２１で実行するので、エミュレータ３９は、ＡｃｔｉｖｅＸコントロールを実行するために（ライン３６３で表された）個々の命令を変換する必要がない。しかしながら、それは、セキュリティを提供する目的のためにそれらをフィルタリングし、変換する。例えば、ＡＰＩはオペレーティングシステム３５のカーネルを呼び出すために、ｘ８６割り込み（ＩＮＴ）命令を使用する。それ故、コントロールにおけるＩＮＴ命令は、ＡＰＩサンク３９１－３９３及びスニフコード３９４をバイパスすることができ、カーネルを直接呼び出すことができる。それ故、エミュレータ３９は、この命令を無条件にブロックし、それは、ライン３６４に全く出力コードを生成しない。ライン３６３でのサブルーチンコール（ＣＡＬＬ）及びリターン（ＲＥＴ）、無条件／条件付のジャンプ（ＪＭＰ／Ｊｘｘ）のような他の問題のある命令は、サブルーチンコールによってライン３６４に置換され、これらの命令のひとつがシミュレーションされたとき、既にコンパイルされたコードのキャッシュは、コール又はジャンプのキャッシュ内の宛先アドレスを判断するために検索される必要がある。

【００２１】

アプレット３６２からのＡＰＩコールは、ＡＰＩコード３５２－３５４に直接処理されない。むしろ、サンクコード３９１－３９３はそれらをインターセプトし、それらで何をするか決定する。３９１でのようないくつかのコールは、サンク３９１によって対応するＡ

ＰＩ３５２に直接通され、これらのコールは、システムに大混乱をもたらすことはなく、従って、セキュリティリスクが存在しない。３９２のような他のサンクは、その特定のコールの所定の特徴に依存して、それに対応するＡＰＩ３５３にコールを通すかどうか決定し、それをＡＰＩに出す前にコールを修正することができる。３９３のようなあるサンクは、コールをそれらのＡＰＩ３５４に完全に認めず、これらのコールは、システムのセキュリティを犯し、信頼できないアプレット３６２によって許容されない。

【００２２】

図３は、パーソナルコンピュータ２０で実行されるアプレットがパーソナルコンピュータのセキュリティを有しない全てのオペレーティングシステムサービスにアクセスすることができる発明のある実施形態の大まかなステップ４００を図示する。

【００２３】

ステップ４１０では、ウェブブラウザのようなホストアプリケーションがアプレットを割り当てられたメモリ範囲にロードする。割り当てられたメモリ範囲を、このアプリケーションではサンドボックスと呼ぶ。サンドボックスは、アプレットをストアするための最初のメモリセグメントと、アプレットを実行する間、ストレジをアドレス可能にするためのランタイムメモリセグメントとの両方を含み、これらは在来のいかなる手段でも割り当てられ得る。この実施形態では、ＯＳ３５は、ステップ４１１でエミュレータ３９を呼び出す。ステップ４１２は、アプレット３６２のコードをストアするために、図１のＲＡＭ２２におけるアドレスの領域及び範囲を割り当て、ランタイムワーキングストレージを使用するためのアプレットに関する他の領域を割り当て、これらの２つの領域は、他のいかなるアプレット、アプリケーションプログラム、又は他のシステムのファシリティに影響を与えることなく、安全に実行することができるサンドボックスを一緒に構成する。それらは、各セキュリティドメインのためのひとつのＸＷ８６サンドボックスとなる、即ち同じセキュリティ設定を有する全てのコントロールが同じサンドボックスでプレイする。セキュリティ設定がウェブページのＵＲＬ（ｕｎｉｆｏｒｍ ｒｅｓｏｕｒｃｅ ｌｏｃａｔｏｒ）を含むので、各オープンウェブページは、少なくとも１つのサンドボックスを有する。通常、同じウェブページの全てのコントローラは、同じサンドボックスにある。それらのカスタムインターフェースが安全でないけれども、サンドボックス内でインターアプレットを実行することは許容される。

【００２４】

ステップ４２０は、実行のためにアプレットを準備する。

【００２５】

ステップ４２１は、リンクを備えるアプレットの静的リンクをサンクモジュールと置換する。即ち、エミュレータ３９は、アプレット３６２のコード内でＡＰＩ３５２－３５４に対する全てのコールを見つけ、それらを対応するサンク３９１－３９３に対するコールに変更する。静的リンクは、アプレットの実行中、一定を維持するリンクである。ＤＬＬ即ちダイナミックリンクライブラリは、実行可能な関数のライブラリ、又は、Ｗｉｎｄｏｗｓアプリケーションによって使用することができるデータである。典型的には、ＤＬＬは、１又はそれ以上の特定の関数を提供し、ＤＬＬは、ＤＬＬに対する静的又は動的なリンクのいずれかを生成することによってアクセスされる。ＤＬＬは、最後に拡張子．ｄｌｌを備える記述でファイルされる。サンクＤＬＬは、サンドボックス内の安全なＡＰＩである。サンクＤＬＬは、安全であると考えられない多くのＡＰＩをブロックし、制限する。例えば、ＣｒｅａｔｅＦｉｌｅが知られたロケーションにだけ許容されうる。同様に、アプレットは、パスワードを記録するための他の処理を生成することができない。上述のように、いくつかのサンクは、対応するＡＰＩにコントロールを単に通す。例えば、「ＣｒｅａｔｅＷｉｎｄｏｗ」、「ＣｒｅａｔｅＤｉａｌｏｇ」、「ＣｒｅａｔｅＩｃｏｎ」、「ＣｒｅａｔｅＣｕｒｓｏｒ」と名付けられたＷｉｎ３２ＡＰＩ及び同様な関数は、他のプロセスに影響せず、信頼できないコードを許容しうる。一方、所定の他のＡＰＩは、信頼できないコードを完全に利用できなくさせなければならない。例えば、「ＣｒｅａｔｅＰｒｏｃｅｓｓ」を許容することにより、信頼できないアプレットをサンドボックスの外

側で別のプログラムを実行することができ、「ＥｘｉｔＷｉｎｄｏｗｓＥｘ（）」のようなオペレーションを完全にブロックすることができ、それにより、信頼できないコードは現在のユーザをログオフすることができず、コンピュータをオフにすることができない。３９３のようなサンクが、ライン３９５によって表示されたコントロールにエラーコードバックを戻すことによってＡＰＩをブロックする。

【００２６】

いくつかのＡＰＩはある条件下、又は所定の修正で許容されうる。この場合、３９２のようなサンクは、それが対応するＡＰＩ３５３を呼び出すか又はブロックするかのいずれかであった後、内部演算を実行し、修正されたパラメータをＡＰＩに通す。例えば、「ＳｅｎｄＭｅｓｓａｇｅ（）」は通常メッセージをウィンドウに送信する。ＳｅｎｄＭｅｓｓａｇｅサンクにより、ＡｃｔｉｖｅＸコントロールがメッセージをそのコントロールによって生成されたウィンドウに送信することができる。しかしながら、サンクは、ウェブブラウザによって、又は他のアプリケーションプログラムによるそれ自身の全てのメッセージをブロックする。このことにより、コントロールが、他のプログラムに属するウィンドウによって実行されるべきであるキーストロークをまねるためにＶＭ＿ＣＨＡＲメッセージを送信することによってセキュリティを侵害することを防止する。

【００２７】

他の例は、メモリをどんな場所にも普通に割り当てる「ＧｌｏｂａｌＡｌｌｏｃ」、「ＨｅａｐＣｒｅａｔｅ」のようなＷｉｎ３２ＡＰＩを含む。これらのＡＰＩに関するサンクは、対応するＡＰＩの全体のコードを組み込み、サンドボックスメモリ内で完全に実行するようにリコンパイルし、サンドボックスの境界内のみでメモリを割り当てることができる。

【００２８】

次いで、ステップ４２２は、アプレットのコードを図２のエミュレータ３９によって実行されうるオブジェクトコードにコンパイルする。コードが要求されたものになったとき、コンパイルは直ちに又はパート毎に全て進行し、コンパイルされたコードは、サンドボックスの外側に配置された図４のコンパイルされたキャッシュ３５７に配置される。これらの方法におけるコンパイルは、在来のものであり、発明の本質には関係しない。

【００２９】

ステップ４２３は、認められないメモリリファレンスに対する禁止を強化するためにアプレット自身のコードにチェックコードを挿入する。「スニフ（ｓｎｉｆｆ）コード」と呼ばれるこのチェックコードは、アプレットのコードによって全てのメモリの読み書きを調べ、その結果からそれらを許可し、又は許可しない。アプレットがサンドボックスの外側のメモリにアクセスするのを防止することにより、アプレットのセキュリティは向上する。予め割り当てられた範囲からだけのアプレットに対して全てのメモリを提供することにより、スニフコードオーバーヘッドを低減させ、その結果、メモリ範囲の効率的なチェックを生じる。更なる最適化技術が、基本のブロックレベルでコードをコンパイルすることによって追加される。例えば、種々のメモリリファレンスが同じレジスタを使用するアプレットによってなされるならば、コンパイラは、各アクセスに関するスニフコードに対する別々のコールを生成するのではなく、一回だけそのレジスタによってアドレス可能な全体の範囲をチェックすることができる。詳細な例を図４と一緒に示す。基本的には、スニフコードによって、割り当てられたサンドボックス内と、システムを損傷しない所定の他のメモリ内とだけで、アプレットがＲＡＭアドレスを参照することができる。（エミュレータ３９がサンドボックスの外側でメモリを参照することができないが、それはサンドボックスにメモリ領域を割り当てるための能力を有する。デバイス独立ビットマップイメージのような目的に関して、余分のスニフコードオーバーヘッドは労力より小さく、さもなければ最初のサンドボックス領域内にイメージをコピーするように要求される。）

ステップ４３０は、アプレットを実行する。ステップ４３１は、命令シーケンスに続く。

【００３０】

現在の命令がＡＰＩに対するコールであるならば、ステップ４２１によって配置されたリ

ンクは、コールがステップ４３２で完全にブロックされ、ステップ４３３で実行され、ステップ４３４で更に処理され、次いで、ブロックされ又は許可されうるかどうか判断する。

【００３１】

現在の命令が、ＬＯＡＤ又はＳＴＯＲＥのようなメモリリファレンス命令であるならば、ステップ４３５によって、命令がそのサンドボックス内のアドレスを参照するならば、ステップ４３６がその命令を実行することができる。もしそうでなければ、ステップ４３７は、リファレンスがさもなければ許容されるかどうか判断する。もしそうならば、ステップ４３５はそれを実行し、さもなければ、ステップ４３８はアクセスをブロックし、エラーを返す。スニフコードはこれらのステップを実行する。他のＸ８６命令は、ステップ４３６によって直接実行される。各命令の後、コントロールはステップ４３１に戻る。プロセス４００は、ホストアプリケーションがそれを終了するまで、続く。

【００３２】

いくつかのシステムでは、４３３のようなブロックによってＡＰＩの実行が、他のセキュリティ暴露を現す。ＡＰＩの引き数がサンドボックスにおけるデータに対するポインタであるならば、サンクがＡＰＩに対して示されたメモリのコンテンツ及び、ＡＰＩに対する実際のコールを検査する時間の間は、短い時間である。マルチスレッドアプレットでは、アプレット内の他の実行スレッドが、ＡＰＩに対して示されたメモリのコンテンツを変更し、それによってＡＰＩに対して無効にされたデータを転送することができる。かかるアタックを防止するために、ブロック４３３－１は、ＡＰＩの引数の「ディープコピー（ｄｅｅｐ　ｃｏｐｙ）」を実行し、ブロック４３３－２は、ＡＰＩからの戻り値をディープコピーする。更に特別に、ステップ４３３がＡＰＩを実行するとき、ステップ４３３－１は、ＡＰＩが実際にコールされる前に、サンドボックス内のそれらの位置から、サンドボックスの外の別の位置に、ＡＰＩに通された全ての引数を最初にコピーする。アプレット自身がこのコピーにアクセスすることができないので、ＡＰＩは、既に保存されているデータだけを有効にする。ステップ４３３は、次いで、サンドボックスの外に、戻り値を置き、ＡＰＩコンポーネントを実行した後、ステップ４３３－２は、アプレットの使用のためにサンドボックスの内側に戻り値をコピーする。所望ならば、ディープコピーが、選択的に使用されうる。

【００３３】

図４は、本発明に関するそれらの領域のみを示すシステムＲＡＭ２５のメモリマップである。予め割り当てられた範囲２５１は、サンドボックスを形成する。それは、アプレット３６２と、アプレットの実行中に、アドレス可能なワーキングストレージに関するランタイムメモリセグメント２５２と、変換コードサンク３９１－３９３（ここではサンク３９１としてだけ示す）をストアするためのセグメントと、をストアするための最初のメモリセグメントを包含する。サンドボックス２５１の外側のメモリ２２は、ここでは３５２によって表されるＡＰＩ　ＤＬＬと、カーネル３２３５５とを包含する。他のワーキングメモリ領域は、３５６として表される。コンパイルされたキャッシュ３５７はまた、サンドボックス２１５の外側に配置される。セキュリティポリシーが実際に実行されることがここであるので、サンドボックス２１５の外側のＷＨＫＲＮＬ３２３２５の位置は、特に重要であり、それがサンドボックスの内側にあるならば、ルージュアプレットが、それを修正することによってセキュリティを妥協することができうる。

【００３４】

以下の例は本発明の作動の例示を示す。前に述べたように、この実施例は、ｘ８６Ｗｉｎ３２アプレットを実行するための前述のＷｘ８６ＶＭエミュレータを利用し、Ｗｉｎｄｏｗｓ９５又はＷｉｎｄｏｗｓＮＴオペレーティングシステム下でｘ８６プラットフォームで修正されずにコントロールする。

【００３５】

マイクロソフトインターネットエクスプローラのようなウェブブラウザは、インターネットからハードドライブのｃ：￥ｔｅｍｐ￥ｆｏｏ．ｏｃｘに、ｆｏｏ．ｏｃｘと呼ばれる

ＡｃｔｉｖｅＸコントロール（アプレット）をダウンロードする。拡張子．ｏｃｘはＡｃｔｉｖｅＸコントロールを示す。

【００３６】

次いで、インターネットエクスプローラは、システムにおいてＷｘ８６ＶＭの存在を探す。Ｗｘ８６ＶＭコンポーネントが利用可能であるならば、インターネットエクスプローラは、それを呼び出し、コントロールに関する全てのセキュリティに関する情報を提供し、ロードされるべきコントロールを要求する。Ｗｘ８６ＶＭコンポーネントは、インターネットエクスプローラがそれを提供し、Ｗｘ８６ＶＭにそれを送り出すか、オブジェクトリンカをＯＬＥ３２に行かせるかどうか判断するセキュリティ情報を調べ、それを取り扱う。

【００３７】

コントロールがＷｘ８６ＶＭエミュレータにおいて送り出されるべきならば、Ｗｘ８６ＶＭはメモリの割り当てられた領域、又は、ＡｃｔｉｖｅＸコントロールに関するサンドボックスを生成する。Ｗｘ８６ＶＭは、ＡｃｔｉｖｅＸコントロール（図４では３６２と示す）ｆｏｏ．ｏｃｘをサンドボックス内にロードする。

【００３８】

Ｗｘ８６ＶＭは、３９１のようなＡＰＩサンクＤＬＬ（安全ＡＰＩ）をサンドボックス内にロードする。Ｗｘ８６ＶＭは、前述の出願（ドケット７７７．０１６ＵＳ１）において十分に説明したようなオペレーティングシステムローダ内でＤＬＬの名前を修正することができる。このことにより、Ｗｘ８６ＶＭは、規定を呼び出す際に違いを取り扱うために、ｘ８６イメージと本来のＡＰＩとの間にサンクコードを挿入することができる。再配置するための名前のリストは、レジストリにストアされる。例えば、カーネル３２（図４では３５５）は、（図４ではサンク３９１と示す）ｗｉｋｒｎｌ３２に再配置され、ｕｓｅｒ３２．ｄｌｌはｗｉｕｓｅｒ３２．ｄｌｌに再配置される。ＡＰＩサンクは次の２つのＤＬＬから構成される：１つは、Ｗｘ８６ＶＭ内で稼働する「ｗｉ」という接頭辞が付いたＤＬＬであり、今後は信頼性がなく、他のひとつはＷｘ８６ＶＭの外側で稼働し、「ｗｈ」という接頭辞が付いたＤＬＬであり、安全ポリシーを実行するために信頼される。

【００３９】

「ｗｉ」ＤＬＬは、それらが置換するための本来のＤＬＬとして同じエクスポートを有する。これらのエクスポートは、サンドボックスの外でスイッチングをするためにＷｘ８６ＶＭに対して応答可能であり、次いで、安全モードにおいて適当なサンクを呼び出し、これは更に、そのＡＰＩに関してセキュリティポリシーを実行する。特定のＡＰＩに関するセキュリティがないならば、サンクは単に本来のＡＰＩを呼び出す。「ＢＯＰ」と呼ばれるこのコールは、典型的には、モードスイッチが起こるのを必要とするＷｘ８６．ｄｌｌを合図する無効なｘ８６　ｏｐｃｏｄｅである。ＢＯＰコマンドは、フォーム「ＢＯＰ（ＤＬＬ＃，ＡＰＩ＃）」を有する。Ｗｘ８６ＶＭが、ＤＬＬがサンドボックスのレジスタセット及びスタックにアクセスする、（図４ではｗｈｋｒｎｌ３２３５２のような）「ｗｈ」という接頭辞であるホストサイドサンクＤＬＬにＢＯＰをディスパッチするとき、ＤＬＬは、パラメータをサンドボックスのスタックから本来のスタックにコピーし、ＡＰＩの引数を検査し、コールを作り、サンドボックスのＥＡＸレジスタに戻り値を戻すように移動させることができる。

【００４０】

例えば、ｘ８６アプレット又はコントロールが、ｋｅｒｎｅｌ３２！ＣｒｅａｔｅＦｉｌｅに対する静的なリンクを有するならば、Ｗｘ８６ＶＭはそのリンクを、ｗｉｋｒｎｌ３２！ＣｒｅａｔｅＦｉｌｅと解決する。アプレットがＣｒｅａｔｅＦｉｌｅを呼び出すとき、ｗｉｋｒｎｌ３２！ＣｒｅａｔｅＦｉｌｅは、サンドボックスからネイティブにスイッチするＢＯＰ命令を実行し、Ｗｘ８６ＩＤｉｓｐａｔｃＢｏｐ（）をＷｘ８６ＶＭＩ．ｄｌｌに呼び出す。Ｗｘ８６ＤｉｓｐａｔｃＢｏｐ（）は、コールをｗｈｋｒｎｌ３２！ｗｈＣｒｅａｔｅＦｉｌｅ（）にディスパッチする。その関数は、ネイティブｋｅｒｎｅｌ３２！ＣｒｅａｔｅＦｉｌｅ（）を呼び出し、戻り値をシミュレーションされたＥ

ＡＸレジスタにコピーし、戻る。
【００４１】
Ｗｘ８６ＶＭはまた、エミュレータ３９コードＷｘ８６ｃｐｕ．ｄｌｌをロードする。アプレットの実行中、プロセッサがＢＯＰ命令に出会うとき、エミュレーションは停止する。
【００４２】
アプレットの実行は、必要とされるコードをコンパイルし、コードをコンパイルされたキャッシュに置くことにより始まる。コンパイルされたコードは、メモリ読み書きオペレーションが安全なオペレーションであることを確認するためにそれにおいてスニフチェックを有する。アクセスされるメモリが、所定のサンドボックス領域の外側であるならば、メモリにアクセスする試みのオペレーションは失敗する。例えば、アプレットｆｏｏ．ｏｃｘが命令ＭＯＶ ＥＡＸ，［ＥＳＩ＋４］を包含するならば、コンパイラは、命令が安全であることを確認するためのＭＯＶ命令の前にスニフコードを挿入する。以下の命令：
ＭＯＶ ＥＡＸ，［ＥＳＩ＋４］
は、スニフコードが挿入された後に、
ＬＥＡ ＥＣＸ，［ＥＳＩ＋４］
ＣＡＬＬ ＳＮＩＦＦＲＥＡＤ４．ＥＣＸ
ＭＯＶ ＥＡＸ，［ＥＣＸ］
となる。
【００４３】
スニフコードがオーバーヘッドに加わるので、基本的なブロックレベルでコードをコンパイルするとき、追加の最適化技術は適用されうる。例えば、アプレットが同じレジスタを使用する種々のメモリリファレンスを作るならば、コンパイラは、一回だけ全体の範囲をチェックし、個々のスニフコールを生成しない。アプレットｆｏｏ．ｏｃｘが以下の命令を包含するならば、
ＭＯＶ ＥＡＸ，［ＥＳＩ＋４］
ＭＯＶ ＥＤＸ，［ＥＳＩ＋８］
より効率の低い仕方でスニフコードを次のように挿入するよりも、
ＬＥＡ ＥＡＸ，［ＥＳＩ＋４］
ＣＡＬＬ ＳＮＩＦＦＲＥＡＤ４．ＥＣＸ
ＭＯＶ ＥＡＸ，［ＥＣＸ］
ＬＥＡ ＥＡＸ，［ＥＳＩ＋４］
ＣＡＬＬ ＳＮＩＦＦＲＥＡＤ４．ＥＣＸ
ＭＯＶ ＥＤＸ，［ＥＣＸ］
スニフコードは、以下のように挿入される。
ＬＥＡ ＥＡＸ，［ＥＳＩ＋４］
ＣＡＬＬ ＳＮＩＦＦＲＥＡＤ８．ＥＣＸ
ＭＯＶ ＥＡＸ，［ＥＣＸ］
ＭＯＶ ＥＤＸ，［ＥＳＩ＋４］
上の記述は例示的なものであり、制限的なものではない。上の記述をみれば、当業者にとって多くの他の実施形態が明らかである。それ故、本発明の範囲は、特許請求の範囲と均等な範囲とあわせて、特許請求の範囲を参照して決定されるべきである。
【図面の簡単な説明】
【図１】本発明が実施される、例示的な計算環境のシステム図である。
【図２】本発明を組み込む実行環境のブロック図である。
【図３】本発明の主なステップを記述するフローチャートである。
【図４】メモリ内のサンドボックス領域の簡単化したブロック図である。

【図１】

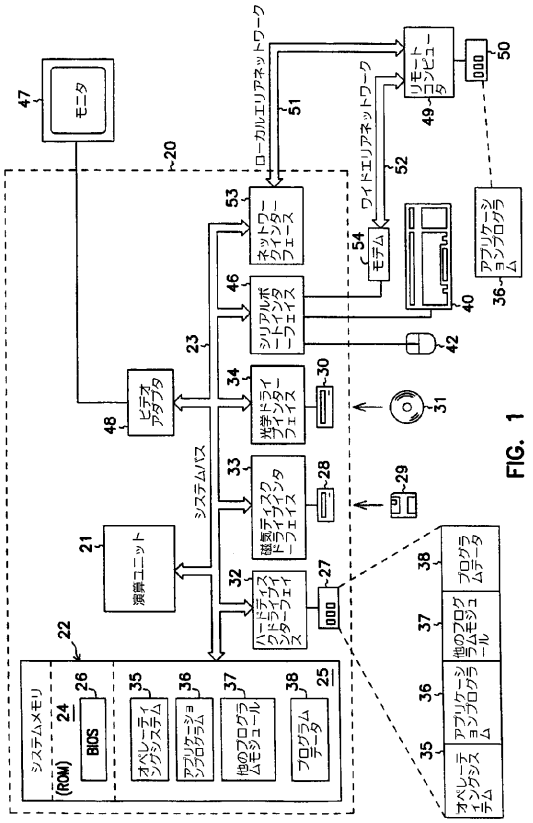

FIG. 1

【図２】

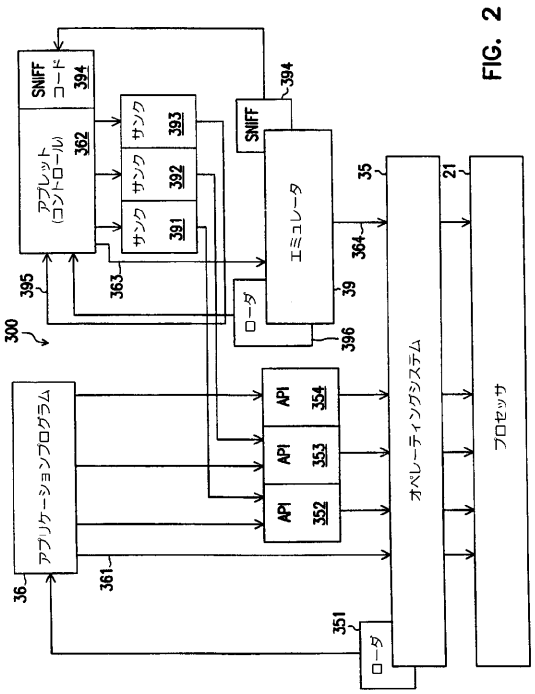

FIG. 2

【図３】

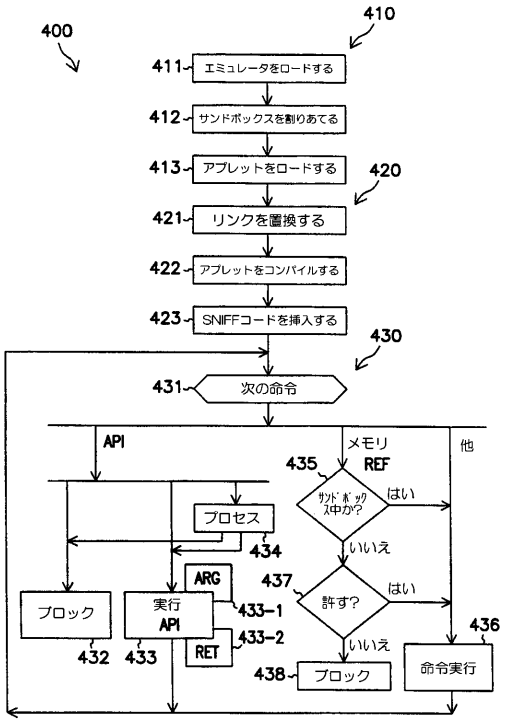

FIG. 3

【図４】

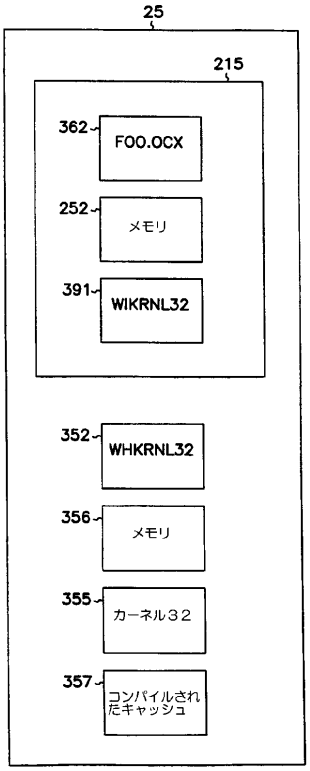

FIG. 4

フロントページの続き

(72)発明者 バラッティー スディープ
アメリカ合衆国 ワシントン州 ９８００８ ベルヴィュー ワンハンドレッドアンドシックスティフィフス プレイス ノースイースト ３２７２

審査官 久保 光宏

(56)参考文献 特開平７－２３０３８０（ＪＰ，Ａ）
特開平３－７８０３１（ＪＰ，Ａ）
米国特許第５３９８１９６（ＵＳ，Ａ）
特開平８－１９４５０４（ＪＰ，Ａ）
特開平１０－１２４３２４（ＪＰ，Ａ）
特開平１０－８３３１０（ＪＰ，Ａ）
特表２０００－５１６７４０（ＪＰ，Ａ）
Hamilton M.A., "Java and the Shift to Net-Centric Computing", Computer, １９９６年 ８月, Vol.29, No.8, pp.31-39
戸松豊和, 「ＪＡＶＡがもたらしたもの １ Ｗｅｂアプレットの誕生」, ｂｉｔ, 日本, 共立出版株式会社, １９９６年 ８月 １日, Vol.28, No.8, pp.26-32
戸松豊和, 「ＪＡＶＡがもたらしたもの ２ Ｊａｖａ仮想機械」, ｂｉｔ, 日本, 共立出版株式会社, １９９６年 ９月 １日, Vol.28,No.9, pp.28-43
松原敦, 「Java人気の沸騰で窮地に立つMicrosoft」, 日経バイト, 日本, 日経ＢＰ社, １９９６年 ３月２２日, No.150, pp.134-143
服部雅幸, 「ＪＡＶＡトレンドが読めるツボ」, 日経MAC, 日本, 日経ＢＰ社, １９９７年 ７月１５日, No.52, pp.164-173
Shoffner, M.・他, 「JavaとWeb実行可能なオブジェクトのセキュリティ」, ＤＤＪ（Dr. Dobb's Journal Japan）, 日本, 株式会社翔泳社, １９９７年 ３月 １日, Vol.6, No.3, pp.50-58
山口浩, 「ネットワークインフラとしてのJava環境」, 電子情報通信学会技術研究報告, 日本, 社団法人電子情報通信学会, １９９７年 ３月２１日, Vol.96, No.607（OFS96-66～73）, pp.7-13
西岡利博・他, 「オブジェクト指向分散環境ＯＺのセキュリティモデル」, 情報処理学会研究報告, 日本, 社団法人情報処理学会, １９９７年 ８月２２日, Vol.97, No.78（97-PRO-14）, pp.103-108
ハーフヒル トム・Ｒ, 「Microsoftの心理とWindows95の実像」, 日経バイト, 日本, 日経ＢＰ社, １９９５年 ９月 １日, No.142, pp.201-216

(58)調査した分野(Int.Cl.[7], ＤＢ名)
G06F9/06,
G06F9/45,
G06F11/00,
G06F12/14,
G06F15/00,
JSTファイル(JOIS),
CSDB(日本国特許庁),
INSPEC(DIALOG)