(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4913353号
(P4913353)

(45)発行日　平成24年4月11日(2012.4.11)　(24)登録日　平成24年1月27日(2012.1.27)

(51)Int.Cl.　　F I
G06F 21/00 (2006.01)　　G06F 21/00　152
G06F 11/28 (2006.01)　　G06F 11/28　310A
G06F 11/34 (2006.01)　　G06F 11/34　S

請求項の数 4　(全 24 頁)

(21)出願番号　特願2005-89891(P2005-89891)
(22)出願日　平成17年3月25日(2005.3.25)
(65)公開番号　特開2006-268775(P2006-268775A)
(43)公開日　平成18年10月5日(2006.10.5)
審査請求日　平成20年3月21日(2008.3.21)

(73)特許権者　392026693
株式会社エヌ・ティ・ティ・ドコモ
東京都千代田区永田町二丁目１１番１号
(74)代理人　100083806
弁理士　三好　秀和
(74)代理人　100100712
弁理士　岩▲崎▼　幸邦
(74)代理人　100095500
弁理士　伊藤　正和
(74)代理人　100101247
弁理士　高橋　俊一
(74)代理人　100117064
弁理士　伊藤　市太郎

最終頁に続く

(54)【発明の名称】ソフトウェア動作モデル化装置及びソフトウェア動作監視装置

(57)【特許請求の範囲】
【請求項１】
監視対象ソフトウェアの正常動作をモデル化するソフトウェア動作モデル化装置と、前記監視対象ソフトウェアの動作を監視するソフトウェア動作監視装置とを含むソフトウェア動作監視システムであって、
前記ソフトウェア動作モデル化装置は、
前記監視対象ソフトウェアを複数回試行させて、前記監視対象ソフトウェアが発行したインストラクションを取得するモデル化装置側インストラクション取得部と、
前記モデル化装置側インストラクション取得部が取得したインストラクションを試行毎に時系列で蓄積するモデル化装置側インストラクション蓄積部と、
前記監視対象ソフトウェアの分岐命令による分岐フローに基づいて生成した木構造モデルである第１の動作モデルを蓄積するモデル化装置側第１のモデル蓄積部と、
前記インストラクションの時系列から求めた共起頻度を特徴量とする動作モデルである第２の動作モデルを蓄積するモデル化装置側第２のモデル蓄積部と、
今回読み込んだインストラクションの時系列を学習系列とし、前記学習系列と前記モデル化装置側第１のモデル蓄積部に蓄積された動作モデルとを時系列で比較し、前記動作モデルと異なるインストラクションが学習系列上に現れた時点で、前記動作モデルに新たな辺を追加することで木構造モデルを生成し、当該木構造モデルを新たな第１の動作モデルとして前記モデル化装置側第１のモデル蓄積部に蓄積する第１のモデル生成部と、
前記学習系列から導出した共起頻度と、前記モデル化装置側第２のモデル蓄積部に蓄積

された前記第２の動作モデルとから新たな共起頻度を導出し、当該共起頻度を特徴量とする動作モデルを生成し、当該動作モデルを新たな第２の動作モデルとして前記モデル化装置側第２のモデル蓄積部に蓄積する第２のモデル生成部と
を備え、
前記ソフトウェア動作監視装置は、
前記監視対象ソフトウェアが動作中に発行したインストラクションを取得する監視装置側インストラクション取得部と、
前記監視装置側インストラクション取得部が取得したインストラクションを時系列で蓄積する監視装置側インストラクション蓄積部と、
前記ソフトウェア動作モデル化装置によって生成された前記第１の動作モデルを蓄積する監視装置側第１のモデル蓄積部と、
前記ソフトウェア動作モデル化装置によって生成された前記第２の動作モデルを蓄積する監視装置側第２のモデル蓄積部と、
前記監視装置側第１のモデル蓄積部から前記第１の動作モデルを取得し、前記監視装置側インストラクション取得部がインストラクションを取得する度に、前記第１の動作モデルに基づいてトレースし、乖離した場合に異常であると判定する第１の検証部と、
前記監視装置側インストラクション蓄積部が蓄積したインストラクションの時系列のうち、前記第１の検証部が乖離と判断した時点からの時系列を検証系列とし、前記監視装置側第２のモデル蓄積部から前記第２の動作モデルを取得し、前記検証系列の共起頻度を導出し、前記第２の動作モデルとの判別分析によって、乖離が検出された場合に異常であると判定する第２の検証部と
を備えることを特徴とするソフトウェア動作監視システム。
【請求項２】
前記検証系列は、前記監視対象ソフトウェアが終了するまでに発行されたインストラクションの時系列の部分列の集合であることを特徴とする請求項１に記載のソフトウェア動作監視システム。
【請求項３】
前記部分列は、前記インストラクションの時系列を前記監視対象ソフトウェアがシステムコールを発行するタイミングで区切ることで生成されることを特徴とする請求項２に記載のソフトウェア動作監視システム。
【請求項４】
監視対象ソフトウェアの正常動作をモデル化するとともに、前記監視対象ソフトウェアの動作を監視するソフトウェア動作監視システムであって、
前記監視対象ソフトウェアを複数回試行させて前記監視対象ソフトウェアが発行したインストラクション、及び前記監視対象ソフトウェアが動作中に発行したインストラクションを取得するインストラクション取得部と、
前記取得したインストラクションを時系列で蓄積するインストラクション蓄積部と、
前記監視対象ソフトウェアを複数回試行させて前記監視対象ソフトウェアが発行したインストラクションのうち、前記監視対象ソフトウェアの分岐命令による分岐フローに基づいて生成した木構造モデルである第１の動作モデルを蓄積する第１のモデル蓄積部と、
前記監視対象ソフトウェアを複数回試行させて前記監視対象ソフトウェアが発行した前記インストラクションの時系列から求めた共起頻度を特徴量とする動作モデルである第２の動作モデルを蓄積する第２のモデル蓄積部と、
今回読み込んだインストラクションの時系列を学習系列とし、前記学習系列と前記第１のモデル蓄積部に蓄積された動作モデルとを時系列で比較し、前記動作モデルと異なるインストラクションが学習系列上に現れた時点で、前記動作モデルに新たな辺を追加することで木構造モデルを生成し、当該木構造モデルを新たな第１の動作モデルとして前記第１のモデル蓄積部に蓄積する第１のモデル生成部と、
前記学習系列から導出した共起頻度と、前記第２のモデル蓄積部に蓄積された前記第２の動作モデルとから新たな共起頻度を導出し、当該共起頻度を特徴量とする動作モデルを

生成し、当該動作モデルを新たな第２の動作モデルとして前記第２のモデル蓄積部に蓄積する第２のモデル生成部と、
前記第１のモデル蓄積部から前記第１の動作モデルを取得し、前記監視対象ソフトウェアが動作中に発行したインストラクションを前記インストラクション取得部が取得する度に、前記第１の動作モデルに基づいてトレースし、乖離した場合に異常であると判定する第１の検証部と、
前記監視対象ソフトウェアが動作中に発行したインストラクションであって、前記インストラクション蓄積部が蓄積したインストラクションの時系列のうち、前記第１の検証部が乖離と判断した時点からの時系列を検証系列とし、前記第２のモデル蓄積部から前記第２の動作モデルを取得し、前記検証系列の共起頻度を導出し、前記第２の動作モデルとの判別分析によって、乖離が検出された場合に異常であると判定する第２の検証部と
を備えることを特徴とするソフトウェア動作監視システム。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、ソフトウェア動作モデル化装置及びソフトウェア動作監視装置に関する。
【背景技術】
【０００２】
ＰＣやワークステーション、サーバ、ルータ、携帯電話、ＰＤＡなど、すべての計算機は外部もしくは内部からの攻撃にさらされている。代表的な攻撃は、計算機上で実行されているソフトウェアの脆弱性を踏み台にしたものである。攻撃者はソフトウェアの脆弱性を利用した悪意のある実行コードを計算機に送り込み、実行中のプロセスの制御を奪い、当該プロセスの権限を利用して不正操作をおこなう。ソフトウェアの脆弱性を利用した攻撃の対策として、実行中ソフトウェアの異常を検知するシステムが開示されている（例えば、非特許文献１及び非特許文献２参照。）。
【０００３】
非特許文献１に記載のシステムは、ソフトウェアのソースコードを予め取得しておき、ソースコードを静的解析することによってソフトウェアの正常な動作、すなわち、正常動作時に発行されるシステムコールの発行パターンを表すモデルを作成し、ソフトウェアの実行系列がこのモデルに受理されるどうかを検査することによって、正常動作からの乖離すなわち異常動作を検知するものである。モデルの生成はコンパイラによって行われ、モデルが決定性有限オートマトンで記述されることがこのシステムの特徴である。ソフトウェアのソースコードには条件分岐や関数呼び出しなど非決定性を生む箇所が多数存在するため、静的解析からは非決定性オートマトンのクラスでしかモデルの生成ができないが、関数ごとに生成されたオートマトンの直列接続および独自システムコールの追加および修正を行うことで非決定性を削減している。又、関数ポインタやlongjmp命令、シグナルは、性質上静的解析によってオートマトンに反映させることが不可能であるとされるが、監視対象ソフトウェアを動作させながら、モデルへ動的に反映させることで対応していることも特徴である。
【０００４】
非特許文献２に記載のシステムも、非特許文献１同様ソフトウェアの正常な動作、すなわち、システムコールの発行パターンを表すモデルを作成するが、ソースコードを利用せず、攻撃から隔離された環境下でソフトウェアを動作させ、そこで得られる入出力などのログを取得しモデルを学習するものである。システムコール、コールスタックの状況、プログラムカウンタを入力系列とし、プログラムカウンタを状態、システムコール、コールスタックの状況を辺とした非決定性有限オートマトンを学習により生成することが特徴である。システムコールのプログラムカウンタは学習時と検証時に変わってしまう可能性がある（例：動的リンクライブラリ利用時）が、システムコールのプログラムカウンタを直近の静的リンク関数のアドレスとすることで、検証時に、学習したオートマトンを利用できるようにしている。

【０００５】
又、正常動作で発行されるインストラクション集合をあらかじめ設定しておき、コンピュータの動作中に取得したインストラクションが、あらかじめ設定してあった正常動作で発行されるインストラクション集合の中に含まれているか否かを検証するコンピュータ作動状況監視装置も開示されている（例えば、特許文献１参照。）。これは、正常動作では発行されるはずのないインストラクションが発行された場合は、異常であると判定するものである。
【特許文献１】特開平１１－２１９３０４号公報
【非特許文献１】L. Lamら “Automatic Extraction of Accurate Application-Specific Sandboxing Policy”, EUROCOM International Symposium on Recent Advances in Intrusion Detection (RAID) 2004, September 2004
【非特許文献２】D.Gaoら “On Gray-Box Program Tracking for Anomaly Detection”, 13th USENIX Security Symposium, August 2004
【発明の開示】
【発明が解決しようとする課題】
【０００６】
非特許文献１に記載のシステムでは、ソフトウェアのソースコードを予め取得しておかなければならないが、ソフトウェア提供者はソフトウェアの不正な流用を防ぐためにソースコードを開示しないことが容易に考えられる。ソフトウェア提供者がソースコードを静的に解析し生成したモデルを端末側に送ることも考えられるが、モデルはソースコードとほぼ同等の意味をもつため、ソースコードを開示したこととほぼ同じとなる。そのため、ソフトウェア提供者はモデルの開示さえしないと考えられる。又、このシステムは条件分岐や関数ポインタなど実行コードの制御が移り変わる箇所の検証を考慮しているが、条件分岐がソースコードどおりに動作しているかを検証していない。
【０００７】
又、非特許文献２に記載のシステムでは、システムコールのプログラムカウンタではない値をシステムコールのプログラムカウンタとしている。そのため、攻撃者はプログラムカウンタの値を偽装し、モデルに受理される攻撃コードを生成することが可能である。攻撃者がこの偽装を行うためには、モデルに受理されるプログラムカウンタの値を知る必要がある。そのため、偽装から守るためにプログラムカウンタをランダム化し、プログラムカウンタの値を秘匿化することが考えられるが、完全に秘匿化するためにはソースコードが必要となってしまう。以上により、非特許文献１に記載のシステムと同様の理由で、適用することが難しい。又、モデルがプログラムカウンタを状態とするオートマトンで記述される。オートマトンは過去の検証結果を現在の検証に利用しない場合に効率的なモデルであるが、プログラムカウンタを状態としているため、条件分岐などによって入り線が複数の状態が存在することになり、条件分岐を正しく検証しようとすると、過去の条件分岐の結果を保持する必要があるため、非効率である。
【０００８】
更に、非特許文献１、非特許文献２に記載のシステムはともに、システムコールの発生パターンをモデルとしているが、システムコールの発行は、ソフトウェアの動作の一部をモデル化しているにすぎない。そのため、攻撃者はバッファーオーバーフローなどを用いて、モデルに受理されるようにプログラムカウンタやリターンアドレスを偽装し、システムコールの発行を行い、かつ悪意ある実行コードを実行することが可能であった（偽装攻撃）。
【０００９】
又、特許文献１に記載のコンピュータ作動状況監視装置は、システムコールよりも詳細なデータであるインストラクションを利用した監視を行っているが、インストラクションの時系列を考慮していないため、正常動作中に発行されるインストラクションのみを用いて攻撃コードを記述することが容易にできてしまう。したがって、攻撃を確実に検知できるとはいえない。

【００１０】
又、実際のソフトウェアの動作から発行されるシステムコールやインストラクションからモデルを学習する、非特許文献２、特許文献１に記載の技術は、監視対象ソフトウェアの動作すべてを学習の段階で試行しなければならないが、データ領域に格納されたデータを含めて試行することや、すべての動作の組み合わせを試行することは一般的に不可能である。そのため、ソフトウェアの動作検証時に、本来ソフトウェアの正常動作であるが未学習であるため、正常動作ではないと判断せざるを得ない（誤検出）。
【００１１】
そこで、本発明は、上記の課題に鑑み、偽装攻撃を防止するとともに、未学習動作が検知された場合の善悪判断を行うことができるソフトウェア動作モデル化装置及びソフトウェア動作監視装置を提供することを目的とする。
【課題を解決するための手段】
【００１２】
上記目的を達成するため、本発明の第１の特徴は、監視対象ソフトウェアの正常動作をモデル化するソフトウェア動作モデル化装置であって、監視対象ソフトウェアを複数回試行させて、（ａ）監視対象ソフトウェアが動作中に発行したインストラクションを取得するインストラクション取得部と、（ｂ）取得したインストラクションを試行毎に時系列で蓄積するインストラクション蓄積部と、（ｃ）インストラクション蓄積部に蓄積されたインストラクションの時系列を試行毎に読み込み、インストラクションの時系列から動作モデルを作成するモデル生成部と、（ｄ）モデル生成部が前回の読み込みまでで生成した監視対象ソフトウェアの動作モデルを蓄積するモデル蓄積部とを備え、モデル蓄積部は、（ｅ）モデル生成部が前回の読み込みまでで生成した木構造モデルを蓄積する第１のモデル蓄積部と、（ｆ）モデル生成部が前回の読み込みまでで生成したインストラクション発行の統計量を特徴量とする動作モデルを蓄積する第２のモデル蓄積部とを有し、モデル生成部は、（ｇ）今回読み込んだインストラクションの時系列を学習系列とし、学習系列と第１のモデル蓄積部に蓄積された動作モデルとから木構造モデルを生成する第１のモデル生成部と、（ｈ）学習系列から統計量を導出し、更に、第２のモデル蓄積部に蓄積された動作モデルを、統計量を用いて学習する第２のモデル生成部とを有するソフトウェア動作モデル化装置であることを要旨とする。
【００１３】
第１の特徴に係るソフトウェア動作モデル化装置によると、システムコールよりも詳細なデータであるインストラクションの時系列を利用した監視を行うことができ、偽装攻撃をほぼ不可能にできるモデルを生成できる。かつ、木構造モデルの各点は、プログラムカウンタなど入り線を複数にする情報を含めないため、過去の条件分岐の結果はモデルが保持していることとなる。すなわち、条件分岐を正しく検証することが可能なモデルを生成できる。
【００１４】
又、木構造モデルだけでは未学習動作を異常動作として判定せざるを得ないが、インストラクション発行の統計量を特徴とした動作モデルを、木構造モデル生成と同時に生成することができる。
【００１５】
又、第１の特徴に係るソフトウェア動作モデル化装置において、第２のモデル蓄積部は、モデル生成部が前回の読み込みまでで生成したインストラクションの共起頻度を特徴量とする動作モデルを蓄積し、第２のモデル生成部は、学習系列から統計量を導出し、更に、第２のモデル蓄積部に蓄積された動作モデルを、共起頻度を用いて学習してもよい。
【００１６】
ソフトウェアの脆弱性をつき攻撃を行うためには、脆弱性をつく攻撃コードを注入する必要があるが、この攻撃コードは監視対象プログラムにはそもそも存在しなかった可能性が高く、統計的に正常挙動との乖離が考えられる。一方、本来正常動作であるが未学習である挙動の場合は、正常動作である以上、監視対象プログラムに存在していたルーチンや

関数を呼び出すわけであり、統計的に正常挙動と近いものが得られる可能性が高いといえる。

【００１７】

このソフトウェア動作モデル化装置によると、正常状態のインストラクションの共起頻度を把握することができ、ソフトウェアの動作の変化をとらえることのできるモデルを生成することができる。動作の検証をする際、オートマトンや木構造モデルによる検証では未学習の動作を検知したら異常と判断せざるを得なかったが、インストラクションの共起頻度を利用するため、統計的に善悪の判断をすることが可能である。

【００１８】

又、第１の特徴に係るソフトウェア動作モデル化装置において、監視対象ソフトウェアの異常動作が入力された場合、取得したインストラクションから、異常動作によって生じたインストラクションの時系列を分離する負例分離部を更に備え、負例分離部は、取得したインストラクションを、第１のモデル蓄積部に蓄積された動作モデル上でトレースし、動作モデルと異なるインストラクションが現れた時点からのインストラクション時系列を異常動作によって生じたインストラクション列である負例であると判断し、第２のモデル生成部は、負例を用いて、異常動作のモデルを生成してもよい。

【００１９】

統計的なモデルで判別分析を行う際、負例を学習することで、正確な判別を行うことが可能である。しかし、負例を入力するためには、異常状態のソフトウェアの動作から、異常動作によって生じたインストラクションを抽出する必要がある。

【００２０】

このソフトウェア動作モデル化装置によると、正例との差分をとることができ、負例を抽出することができる。

【００２１】

又、第１の特徴に係るソフトウェア動作モデル化装置は、監視対象ソフトウェアを動作させた計算機環境情報を、生成した動作モデルにメタ情報として付与するメタ情報付与部を更に備えてもよい。

【００２２】

インストラクションは計算機環境によって異なる。そのため、ソフトウェアの動作を検証する際には、計算機環境が同一のモデルを選択する、もしくは、検証時の環境にモデル上のインストラクションを変更する必要がある。

【００２３】

このソフトウェア動作モデル化装置によると、動作モデルに計算機環境情報を付与することができ、検証時におけるモデル選択や、インストラクションの変更を可能にする。

【００２４】

本発明の第２の特徴は、監視対象ソフトウェアの動作を監視するソフトウェア動作監視装置であって、（ａ）監視対象ソフトウェアが動作中に発行したインストラクションを取得するインストラクション取得部と、（ｂ）取得したインストラクションを時系列で蓄積するインストラクション蓄積部と、（ｃ）監視対象ソフトウェアの木構造モデルを蓄積した第１のモデル蓄積部から木構造モデルを取得し、インストラクション取得部がインストラクションを取得する度に、木構造モデル上でトレースすることで、正常動作との乖離を判定する第１の検証部と、（ｄ）インストラクション蓄積部が蓄積したインストラクションの時系列のうち、第１の検証部が乖離と判断した時点からの時系列を検証系列とし、監視対象ソフトウェアのインストラクション発行の統計量を特徴量とした動作モデルを蓄積した第２のモデル蓄積部から動作モデルを取得し、検証系列からの統計量を生成し、動作モデルとの判別分析によって、正常動作との乖離を判定する第２の検証部とを備えるソフトウェア動作監視装置であることを要旨とする。

【００２５】

第２の特徴に係るソフトウェア動作監視装置によると、インストラクションをひとつずつ監視していくことで正常状態を確実に判別できる木構造モデルと、インストラクション

の時系列を監視することで未学習動作と異常動作を判別できる統計モデルとを利用することができる。 木構造モデルを利用して監視している間、統計モデルを生成、取得する必要がなくなるため、効率がよい。また、未学習の動作の善悪判断が可能になる。

【００２６】

又、第２の特徴に係るソフトウェア動作監視装置において、第２のモデル蓄積部は、監視対象ソフトウェアが発行したインストラクションの共起頻度を特徴量とする動作モデルを蓄積し、第２のモデル検証部は、検証系列からインストラクションの共起頻度を導出し、動作モデルとの判別分析によって、正常動作との乖離を判定してもよい。

【００２７】

このソフトウェア動作監視装置によると、動作の検証をする際、オートマトンや木構造モデルによる検証では未学習の動作を検知したら異常と判断せざるを得なかったが、インストラクションの共起頻度を利用した善悪判断をすることが可能となる。

【００２８】

又、第２の特徴に係るソフトウェア動作監視装置において、インストラクション蓄積部が蓄積したインストラクションの時系列を学習系列とし、第１のモデル蓄積部に蓄積された木構造モデルと、第２のモデル蓄積部に蓄積されたインストラクション発行の統計量を特徴量とする動作モデルとを、学習系列を利用して学習するモデル学習部を更に備え、モデル学習部は、第２の検証部が、監視対象ソフトウェアの動作が正常であると判定した場合のみ、モデルを学習してもよい。

【００２９】

統計量を利用した動作モデルは、インストラクションの時系列から統計量を計算するコストが発生するため、できる限り利用しないようにしたほうがよい。

【００３０】

このソフトウェア動作監視装置によると、未学習動作を木構造モデルへ反映することができるため、次回の監視が効率よく行える。

【００３１】

又、第２の特徴に係るソフトウェア動作監視装置において、検証系列は、監視対象ソフトウェアが終了するまでに発行されたインストラクションの時系列の部分列の集合であってもよい。又、この部分列は、インストラクションの時系列を監視対象ソフトウェアがシステムコールを発行するタイミングで区切ることで生成されてもよい。

【００３２】

第２の検証部で統計量を計算することになるが、終了までのインストラクションの列を統計量計算の対象とするよりも、サブシーケンスを対象としたほうが、異常検知を早く行うことが可能であり、かつ、シーケンスに攻撃コードが含まれる場合には、統計量を計算するシーケンスのうち攻撃コードが閉める割合が大きくなり、より正常動作との乖離を判定しやすくなる。サブシーケンスの終了を決定するトリガは、あらかじめ与えた定数分のインストラクションが発行される、システムコールが発行される、jmp、call、retなど現在動作しているアドレスから離れたアドレスへ動作を遷移させるインストラクションが発行される、などである。特にシステムコールを利用したものは、システムコールがどのようなインストラクション発行を経て発行されたのかを検査するよいタイミングである。攻撃者がプログラムの脆弱性をついてシステムになんらかの影響を与えるためには、システムコールを発行しなければならないが、システムコールの発行タイミングでサブシーケンスを区切ることで、脆弱性をつく攻撃コードを検知できる可能性が高くなる。

【発明の効果】

【００３３】

本発明によると、偽装攻撃を防止するとともに、未学習動作が検知された場合の善悪判断を行うことができるソフトウェア動作モデル化装置及びソフトウェア動作監視装置を提供することができる。

【発明を実施するための最良の形態】

【００３４】

次に、図面を参照して、本発明の実施の形態を説明する。以下の図面の記載において、同一又は類似の部分には、同一又は類似の符号を付している。但し、図面は模式的なものであることに留意すべきである。

【００３５】

＜第１の実施の形態＞

第１の実施の形態に係るソフトウェア動作モデル化装置１００は、図１に示すように、監視対象ソフトウェア１０を複数回試行させて、監視対象ソフトウェアの動作を表す動作モデルを生成する。又、第１の実施の形態に係るソフトウェア動作監視装置２００は、図２に示すように、ソフトウェア動作モデル化装置１００が生成した動作モデルを利用して監視対象ソフトウェア１０の動作を監視し、異常動作がないかを検証する。

【００３６】

（ソフトウェア動作モデル化装置）

ソフトウェア動作モデル化装置１００は、図１に示すように、監視対象ソフトウェア１０が動作中に発行するインストラクションを取得するインストラクション取得部１１０と、取得したインストラクションの時系列を蓄積するインストラクション蓄積部１２０と、インストラクション蓄積部１２０に蓄積されたインストラクション時系列を学習系列とみなし、動作モデルを生成するモデル生成部１３０と、モデル生成部１３０によって生成された木構造モデルを蓄積する第１のモデル蓄積部１４０と、モデル生成部１３０によって生成された共起頻度モデルを蓄積する第２のモデル蓄積部１５０とを備える。モデル生成部１３０は、インストラクション蓄積部１２０に蓄積されたインストラクション時系列より木構造モデルを生成する第１のモデル生成部１３１と、共起頻度モデルを生成する第２のモデル生成部１３２とを有する。

【００３７】

インストラクション取得部１１０は、LinuxのPTRACEを利用するなどして監視対象ソフトウェアがインストラクションを発行するたびにインストラクションを取得する。

【００３８】

インストラクション蓄積部１２０は、取得したインストラクションを試行毎に時系列で蓄積する。

【００３９】

第１のモデル生成部１３１は、動作モデルと学習系列とを時系列で比較し、動作モデルと異なるインストラクションが学習系列上に現れた時点で、動作モデルに新たな辺を追加することで木構造モデルを生成する。

【００４０】

第１のモデル蓄積部１４０は、第１のモデル生成部１３１が前回の読み込みまでで生成した木構造モデルを蓄積する。

【００４１】

第２のモデル生成部１３２は、学習系列から共起頻度などの統計量を導出し、更に、第２のモデル蓄積部に蓄積された動作モデルを、統計量を用いて学習する。

【００４２】

第２のモデル蓄積部１５０は、第２のモデル生成部１３２が前回の読み込みまでで生成したインストラクション発行の共起頻度などの統計量を特徴量とする動作モデルを蓄積する。

【００４３】

メタ情報付与部１７０は、監視対象ソフトウェア１０を動作させた計算機環境情報を、生成した動作モデルにメタ情報として付与する。

【００４４】

図３に動作モデルのデータ構造の例を示す。動作モデルのデータ構造は、インストラクションテーブル３０、分岐先アドレステーブル４０から構成される。インストラクションテーブル３０は、インストラクションの識別子を記録するインストラクション名列３１と、分岐先アドレステーブルへのポインタを記録する分岐ポインタ列３２とから構成され、

インストラクションの時系列、すなわち、木構造の辺を表す。分岐先アドレステーブル４０は、インストラクションテーブルのアドレスを格納する分岐先アドレス列４１の配列であり、分岐前のインストラクションと、分岐先の辺をリンクする。

【００４５】

（ソフトウェア動作モデル化方法）

次に、第１の実施の形態に係るソフトウェア動作モデル化方法について、図４～７を用いて説明する。図４は、第１のモデル生成部１３１が、インストラクション蓄積部１２０に蓄積されたインストラクション時系列より木構造モデルを生成するフローチャートの例である。木構造モデル化には、モデル化関数、モデル化関数に呼ばれ初期モデルを生成する生成関数、モデル化関数に呼ばれ第１のモデル蓄積部１４０に蓄積された木構造モデルを学習する学習関数が用いられる。

【００４６】

まず、ステップＳ１０１において、モデル化関数は、第１のモデル蓄積部１４０に蓄積された監視対象ソフトウェアの動作モデルを取得する。

【００４７】

次に、ステップＳ１０２において、モデル取得に成功したか否か判断し、成功した場合は、ステップＳ１０３へ進み、学習関数を呼び出し、失敗した場合は、ステップＳ１０４へ進み、生成関数を呼び出す。ステップＳ１０３の学習処理と、ステップＳ１０４の生成処理は、後に詳述する。

【００４８】

次に、ステップＳ１０５において、生成もしくは学習された動作モデルを第１のモデル蓄積部１４０へ蓄積する。この際、インストラクションを取得した際の環境情報（例えば、インテルx86プロセッサ等のプロセッサ情報。プロセッサが変わればインストラクション名も変わる）をモデルへ付与することによって、次回、モデルを学習する際や、検証時に、適したモデルを選択することができる。環境情報の取得とメタ情報の付与は、メタ情報付与部１７０が行う。環境情報の取得は、環境情報を格納している箇所、たとえば、Linuxの場合、/proc/cpuinfoにはCPUの情報がおかれているが、/proc/cpuinfoなどを見るなどして取得する。取得した情報はメタ情報としてモデルへ付与される。

【００４９】

次に、図４のステップＳ１０４の詳細について、図５を用いて説明する。

【００５０】

ステップＳ２０１において、生成関数は、インストラクション蓄積部１２０より監視対象ソフトウェアのインストラクション時系列を取得する。

【００５１】

次に、ステップＳ２０２において、取得に成功したか否かを判断し、取得に失敗した場合、ステップＳ２０３へ進み、エラー処理を行い、関数を終了する。一方、成功した場合、ステップＳ２０４へ進み、インストラクションを時系列の先頭から順に読み込む。

【００５２】

次に、ステップＳ２０５において、インストラクション読み込みが終了した場合、生成関数を終了する。

【００５３】

一方、ステップＳ２０５において、インストラクション読み込みが終了していなければ、ステップＳ２０６において、インストラクションテーブルを作成し、ステップＳ２０７において、インストラクション名をインストラクション名列に、ヌルを分岐ポインタ列へ格納する。そして、ステップＳ２０４へ戻り、次のインストラクションを取得する。

【００５４】

次に、図４のステップＳ１０３の詳細について、図６を用いて説明する。

【００５５】

まず、ステップＳ３０１において、学習関数は、インストラクション蓄積部１２０より監視対象ソフトウェアのインストラクション時系列を取得する。

【００５６】
ステップＳ３０２において、取得に成功したか否か判断し、取得に失敗した場合、ステップＳ３０３へ進み、エラー処理を行い、関数を終了する。一方、成功した場合、ステップＳ３０４へ進み、インストラクションを時系列の先頭から順に読み込む。
【００５７】
次に、インストラクション読み込みが終了したか否か判断し、終了した場合、生成関数を終了する。終了していなければ、ステップＳ３０６へ進み、木をトレースする。木トレースについては、後に詳述する。
【００５８】
次に、ステップＳ３０７において、トレースした結果、「読み込んだインストラクションは存在している」と判断すると、次のインストラクションを取得し、「新しい」と判断した場合、ステップＳ３０８へ進む。そして、ステップＳ３０８において、インストラクションテーブルを生成し、ステップＳ３０９において、分岐先アドレステーブルを生成し、新規に生成したインストラクションテーブルのアドレスを分岐先アドレス列に格納する。そして、ステップＳ３１０において、インストラクション名をインストラクション名列に、分岐先アドレステーブルのアドレスを分岐ポインタ列へ格納し、ステップＳ３０４へ戻り、次のインストラクションを取得する。
【００５９】
次に、図６のステップＳ３０６の詳細について、図７を用いて説明する。
【００６０】
まず、ステップＳ４０１において、現在トレースしているアドレスを示す現在位置を利用して、インストラクションテーブルの内容を確認する。
【００６１】
次に、ステップＳ４０２において、インストラクション名に取得したインストラクションが登録されているかを確認し、登録されていれば（インストラクションが新規でない場合）、ステップＳ４１０へ進み、存在していると判断する。そして、ステップＳ４０５において、現在位置を次のアドレスに更新し、木トレースを終了する。
【００６２】
一方、ステップＳ４０２において、登録されていなければ、分岐ポインタの値を確認し、ヌルであれば、ステップＳ４０４へ進み、新規インストラクションであると判断し、ステップＳ４０５において、現在位置を次のアドレスに更新し木トレースを終了する。
【００６３】
ステップＳ４０３において、ヌルでなければ、ステップＳ４０６において、分岐ポインタの値を利用して分岐先アドレステーブルを引き、分岐先アドレスを順に取得する。
【００６４】
次に、ステップＳ４０７において、取得が成功したか否か判断し、成功した場合は、ステップＳ４０８へ進み、分岐先アドレスが引くインストラクションテーブルの先頭のインストラクション名を確認する。そして、ステップＳ４０９において、取得したインストラクションが登録されているかを確認し、登録されていれば（インストラクションが新規でない場合）、ステップＳ４１０へ進み、存在していると判断する。そして、ステップＳ４０５において、現在位置を次のアドレスに更新し、木トレースを終了する。一方、ステップＳ４０９において、登録されていなければ、ステップＳ４０６へ戻り、次の分岐先アドレスを取得する。
【００６５】
一方、ステップＳ４０７において、取得が失敗した場合、ステップ
Ｓ４０４において、新規インストラクションであると判断し、ステップＳ４０５において、現在位置を次のアドレスに更新し、木トレースを終了する。
【００６６】
次に、図８～１０を用いて、モデルが学習される様子を説明する。
【００６７】

図８は、インストラクション蓄積部１２０に蓄積されたインストラクション時系列である。最左列には通し番号がふられており、１から順に読み込まれ、モデルに変換される。

【００６８】

図９は２番のインストラクション時系列を読み込んで学習が終了した時点のモデルである。時系列上６番目のインストラクションにおいて、１番の時系列ではjmp、２番の時系列ではpopが存在しているため、jmpの分岐ポインタ列には分岐先アドレステーブルへのアドレス（AAAAAAAA）が登録され、分岐先アドレステーブルによって、第２のインストラクションテーブルが引かれるように学習されている。

【００６９】

図１０は、３番のインストラクション時系列を読み込んで学習が終了した時点のモデルである。順次インストラクション時系列が読み込まれるが、３番のインストラクション時系列上の１９番目のインストラクションにおいて、図８のモデルに登録されていない新たなインストラクションmovが存在しているため、jmpの分岐ポインタ列には分岐アドレステーブルへのアドレス（BBBBBBBB）が登録され、分岐先アドレステーブルによって第３のインストラクションテーブルが引かれるよう学習されている。

【００７０】

次に、第１の実施の形態に係るソフトウェア動作モデル化方法について、図１１～図１３を用いて説明する。第２のモデル生成部１３２は、インストラクション蓄積部１２０に蓄積されたインストラクションの時系列からN-gramを生成し、各N-gramの出現確率を動作モデルとして生成することを特徴とする。ここで計算される出現確率は、各N-gramの出現頻度を、生成したすべてのN-gramの出現頻度の総和で除算して求めるが、この分母は、複数回の試行により得られたインストラクションの時系列から生成されたN-gramを総合して求めてもよいし、試行ごと求めてもよい。

【００７１】

図１１に、第２のモデル生成部１３２の動作例を表すフローチャートを示す。第２のモデル生成部１３２は、モデル化関数、生成関数、学習関数からなる。

【００７２】

まず、ステップＳ５０１において、モデル化関数は、第２のモデル蓄積部１５０から監視対象ソフトウェアに対応するモデルを取得する。

【００７３】

そして、ステップＳ５０２において、取得に成功した場合は、ステップＳ５０３へ進み、取得に失敗した場合は、ステップＳ５０４へ進む。ステップＳ５０３及びステップＳ５０４の詳細は、後に詳述する。

【００７４】

次に、ステップＳ５０５において、学習、生成されたモデルを第２のモデル蓄積部１５０に蓄積する。蓄積の際には、インストラクションを取得した際の環境情報をモデルへ付与することによって、次回モデルを学習する際や、検証時に、適したモデルを選択することができる。

【００７５】

次に、図１１のステップＳ５０４の詳細について、図１２を用いて説明する。

【００７６】

まず、ステップＳ６０１において、生成関数では、インストラクション蓄積部１２０からインストラクション時系列を取得する。ステップＳ６０２において、取得に成功した場合、ステップＳ３０４へ進み、N-gramを生成する。そして、ステップＳ３０５において、各N-gramの出現頻度を計算し、生成を終了する。

【００７７】

一方、ステップＳ６０２において、取得失敗した場合、エラー処理後に生成を終了する。

【００７８】

ここで生成されるN-gramは、たとえばひとつのインストラクションを１ワードとした連

続Ｎワードの列であり、たとえば“A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese”に記載のアルゴリズムを用いるなどして生成される。本実施形態では、ひとつのインストラクションを１ワードとして取り扱うが、複数のインストラクションを１ワードとしてもよく、機能を限定するものではない。

【００７９】

次に、図１１のステップＳ５０３の詳細について、図１３を用いて説明する。

【００８０】

まず、ステップＳ７０１において、学習関数では、インストラクション蓄積部１１０からインストラクション時系列を取得し、取得に成功した場合、ステップＳ７０４において、上記の方法を用いるなどしてN-gramを生成し、ステップＳ７０５において、モデルの各N-gramの出現頻度の更新および新しいエントリを追加する。

【００８１】

一方、ステップＳ７０２において、取得に失敗した場合、エラー処理後に生成を終了する。

【００８２】

図１４にN-gramの例を示す。最右に位置する数字が出現頻度である。本実施形態では3-gramを例としているが、Nの数を限定するものではない。

【００８３】

N-gramとは、言語処理の手法の一つで、ストリーム上の連続したＮ個の文字あるいは単語の列であり、軽量なアルゴリズムで文字や単語の共起関係を把握できる、優れた手法である。ソフトウェアとは、ソースコードとソースコードから呼ばれるライブラリ上のバイナリコードで構成される。すなわち、動作の異なるソフトウェアを生成するためには、おおむねソースコードを変更することになる。又、ソフトウェアには、外部からの実行コードの参入を許し、動的に動作を変更できるものも存在するが、ソフトウェア実行中に発行されるインストラクションは、動作の変更にともない変化する。すなわち、どちらの場合においても、動作が変化すれば正常状態とは異なる共起関係を有するインストラクション時系列を発行する。

【００８４】

正常かつ未学習の動作の場合は、同一のソースコードから発行されるインストラクションであれば、モデルと統計的に近いサンプルが得られる可能性が高い。なぜならば、同一の環境下でコンパイラを通って生成されたインストラクションであるからである。攻撃者の作成する攻撃コードは、アセンブラ（又は機械語）でかかれることが多く、ソフトウェアの被攻撃時には、実行コードが発行するインストラクションとは異なる共起関係を有するインストラクション時系列を発行する可能性が高い。

【００８５】

（ソフトウェア動作監視装置）

ソフトウェア動作監視装置２００は、図２に示すように、監視対象ソフトウェア１０を入力とし、ソフトウェア動作モデル化装置１００によって生成された木構造モデルを蓄積する第１のモデル蓄積部２５０と、インストラクション発行の統計量（例えばインストラクションの共起頻度）を特徴とした動作モデルを蓄積する第２のモデル蓄積部２６０と接続する。

【００８６】

又、第１の実施の形態に係るソフトウェア動作監視装置２００は、監視対象ソフトウェア１０が動作中に発行したインストラクションを取得するインストラクション取得部２１０と、取得したインストラクションを時系列で蓄積するインストラクション蓄積部２２０と、インストラクション取得部２１０がインストラクションを取得するたびに第１のモデル蓄積部２５０から取得した木構造モデルで監視対象ソフトウェアの発行するインストラクションを検証する第１の検証部２３０と、インストラクション蓄積部２２０に蓄積したインストラクション時系列のうち、第1の検証部２３０が動作の異常を検知してからのイ

ンストラクションを検証系列としてとらえ、第２のモデル蓄積部２６０から取得したインストラクション発行の統計量（例えばインストラクションの共起頻度）を特徴とした動作モデルで監視対象ソフトウェアの発行するインストラクションを検証する第２の検証部２４０とを備え、第２の検証部２４０は、第1の検証部２３０がソフトウェア動作に異常を検知した場合にのみ動作する。

【００８７】

即ち、第１の検証部２３０は、監視対象ソフトウェアの木構造モデルを蓄積した第１のモデル蓄積部２５０から木構造モデルを取得し、インストラクション取得部２１０がインストラクションを取得する度に、木構造モデル上でトレースすることで、正常動作との乖離を判定する。

【００８８】

又、第２の検証部２４０は、インストラクション蓄積部２２０が蓄積したインストラクションの時系列のうち、第１の検証部２３０が乖離と判断した時点からの時系列を検証系列とし、監視対象ソフトウェアのインストラクション発行の共起頻度などの統計量を特徴量とした動作モデルを蓄積した第２のモデル蓄積部２６０から動作モデルを取得し、検証系列からの統計量を生成し、動作モデルとの判別分析によって、正常動作との乖離を判定する。

【００８９】

モデル学習部２７０は、インストラクション蓄積部２２０が蓄積したインストラクションの時系列を学習系列とし、第１のモデル蓄積部２５０に蓄積された木構造モデルと、第２のモデル蓄積部２６０に蓄積されたインストラクション発行の統計量を特徴量とする動作モデルとを、学習系列を利用して学習する。そして、モデル学習部２７０は、第２の検証部２４０が、監視対象ソフトウェアの動作が正常であると判定した場合のみ、モデルを学習する。

【００９０】

又、検証系列は、監視対象ソフトウェアが終了するまでに発行されたインストラクションの時系列の部分列の集合である。この部分列は、インストラクションの時系列を監視対象ソフトウェアがシステムコールを発行するタイミングで区切ることで生成される。

【００９１】

尚、第１のモデル蓄積部１４０と第1のモデル蓄積部２５０は同一のものでもよい。又、第２のモデル蓄積部１５０と第２のモデル蓄積部２６０は同一のものでもよい。

【００９２】

（ソフトウェア動作監視方法）

次に、第１の実施の形態に係るソフトウェア動作監視方法について、図１５及び図１６を用いて説明する。

【００９３】

図１５は、第１の検証部２３０の動作例を示すフローチャートである

まず、ステップＳ９０１において、監視対象ソフトウェアの動作モデルを第１のモデル蓄積部２５０から取得する。このとき、監視対象ソフトウェアの動作環境を取得しておいて、動作モデルに付与されているメタ情報と突合し、動作環境にあわせたモデルを取得するとよい。

【００９４】

次に、ステップＳ９０２において、インストラクション蓄積部２２０に蓄積されたインストラクションを順次取得する。そして、ステップＳ９０３において、終了したか否か判断し、終了したならば、ステップＳ９０４において、正常と判定して、ステップＳ９０８において、判定結果を出力し、監視を終了する。

【００９５】

一方、ステップＳ９０３において、終了していなければ、ステップＳ９０５において、木トレース（詳細は、図７参照。）を呼び出す。

【００９６】

次に、ステップＳ９０６において、判定結果が新規であるか否か判断し、新規である場合は、ステップＳ９０７において、異常と判定し、ステップＳ９０８において、結果を出力し、監視を終了する。

【００９７】

一方、ステップＳ９０６において、判定結果が新規でない場合、即ち、存在であった場合、ステップＳ９０２へ戻り、次のインストラクションを取得する。

【００９８】

以下、図８～図１０を用いて、監視対象ソフトウェアが監視される様子を説明する。

【００９９】

第１のモデル蓄積部２５０に、モデル生成のときに説明した図９記載のモデルが蓄積されているとし、図８記載のインストラクション時系列の1番がインストラクション蓄積部２２０に蓄積されているとする。

【０１００】

インストラクション時系列の先頭からインストラクションが順次読み込まれ、モデルとの乖離判定を行う。この場合、インストラクション時系列上のすべてのインストラクションが、モデル上に存在している（木トレースの結果が「存在」）ことから、動作検証部は正常である旨の結果を出力する。

【０１０１】

次に、第１のモデル蓄積部２５０に、モデル生成のときに説明した図９記載のモデルが蓄積されているとし、図８記載のインストラクション時系列の３番がインストラクション蓄積部２２０に蓄積されているとする。インストラクション時系列の先頭からインストラクションが順次読み込まれ、モデルとの乖離判定を行う。この場合、19番目のインストラクションにおいて、新規インストラクションmovが検知され、動作検証部は異常である旨の結果を出力する。

【０１０２】

図１６は、第２の検証部２４０の動作例を示すフローチャートである。

【０１０３】

まず、ステップＳ１００１において、第２の検証部２４０は、監視対象ソフトウェアの動作モデルを第２のモデル蓄積部２６０から取得する。このとき、監視対象ソフトウェアの動作環境を取得しておいて、動作モデルに付与されているメタ情報と突合し、動作環境にあわせたモデルを取得するとよい。

【０１０４】

次に、ステップＳ１００２において、インストラクション蓄積部２２０に蓄積されたインストラクション時系列から 検証系列を取得する。検証系列とは異常判定が起きたインストラクションから終了までのインストラクションの列、あるいはそのサブシーケンスである。第２の検証部２４０が統計的な判断を行う際、統計量を計算することになるが、終了までのインストラクションの列を統計量計算の対象とするよりも、サブシーケンスを対象としたほうが、異常検知を早く行うことが可能であり、かつ、シーケンスに攻撃コードが含まれる場合には、統計量を計算するシーケンスのうち攻撃コードが閉める割合が大きくなり、より正常動作との乖離を判定しやすくなる。サブシーケンスの終了を決定するトリガは、あらかじめ与えた定数分のインストラクションが発行される、システムコールが発行される、jmp、call、retなど現在動作しているアドレスから離れたアドレスへ動作を遷移させるインストラクションが発行される、などである。特にシステムコールを利用したものは、システムコールがどのようなインストラクション発行を経て発行されたのかを検査するよいタイミングである。攻撃者がプログラムの脆弱性をついてシステムになんらかの影響を与えるためには、システムコールを発行しなければならないが、システムコールの発行タイミングでサブシーケンスを区切ることで、脆弱性をつく攻撃コードを検知できる可能性が高くなる。ただし、本実施例では、サブシーケンスの場合の説明をするが、機能を限定するものではない。

【０１０５】

次に、ステップＳ１００３において、インストラクションの時系列を取得し、取得が終了したか否か判断し、取得が終了した場合、ステップＳ１００４において、正常動作であると判定する。そして、ステップＳ１００５において、判定結果を出力して監視を終了する。

【０１０６】

一方、ステップＳ１００３において、取得が終了していなければ、ステップＳ１００６において、時系列よりN-gramを生成し、ステップＳ１００７において、各N-gramの出現頻度を計算し、出現確率を導出しておく。

【０１０７】

次に、ステップＳ１００８において、取得していた動作モデルと、先ほど生成したN-gramの出現確率を判別分析にかける。判別分析は、適当に閾値を定めることによる線形判別分析を用いるなどして、外れ値を検知する。

【０１０８】

ステップＳ１００９において、判別分析の結果、正常であるか否か判断し、正常である場合は、ステップＳ１００２へ戻り、次のインストラクション時系列を取得する。

【０１０９】

一方、ステップＳ１００９において、異常であると判定された場合は、ステップＳ１０１０において、異常であると判断し、ステップＳ１０１１において、異常である旨を出力して、監視を終了する。

【０１１０】

（作用及び効果）

第１の実施の形態に係るソフトウェアモデル動作モデル化装置及びソフトウェアモデル化方法によると、木構造モデル化を行うことによって、システムコールよりも詳細なデータであるインストラクションの時系列を利用した監視を行うことができ、偽装攻撃をほぼ不可能にできるモデルを生成できる。かつ、モデルの各状態は、プログラムカウンタなど入り線を複数にする情報を含めないため、過去の条件分岐の結果はモデルが保持していることとなる。すなわち、条件分岐を正しく検証することが可能なモデルを生成できる。

【０１１１】

又、学習系列から統計量を導出し、第２のモデル蓄積部１５０に蓄積された動作モデルを、統計量を用いて学習することによって、正常状態のインストラクションの共起関係を把握することができ、ソフトウェアの動作の変化をとらえることのできるモデルを生成することができる。動作の検証をする際、オートマトンによる検証では未学習の動作を検知したら異常と判断せざるを得なかったが、インストラクションの共起関係を利用するため、統計的に正常と判断することが可能である。

【０１１２】

ただし、本実施例では共起頻度を計測する手法としてN-gramをあげているが、アソシエーションルールなど、共起頻度が計測できればよく、これにこだわるものではない。

【０１１３】

又、第１の実施の形態に係るソフトウェア動作監視装置及びソフトウェア動作監視方法によると、インストラクションの共起頻度を利用した監視を行うことができ、未学習動作の正常、異常判別を確率的に求めることができる。

【０１１４】

又、第１の検証部２３０のあとに第２の検証部２４０を配置することで、インストラクションをひとつずつ監視していくことで正常状態を確実に判別できる木構造モデルと、インストラクションの時系列を監視することで未学習動作と異常動作を判別できるN-gramモデルを利用することができる。木構造モデルを利用して監視している間、N-gramの出現頻度を計算する必要がなくなるため、効率がよい。

【０１１５】

又、ソフトウェア動作監視装置２００は、モデル学習部２７０をさらに配置し、第２の検証部２４０の検証により、ソフトウェア動作が正常であると判定された場合に、第２の

検証部２４０で用いた検証系列を、木構造モデルに追加してもよい。N-gramのような統計量を特徴量とする動作モデルは、統計量を計算するコストが発生するため、計算コストを考慮するとできる限り利用しないようにしたほうがよい。上記の構成をとることによって、未学習動作を木構造モデルへ反映することができるため、次回の監視が効率よく行える。

【０１１６】

＜第２の実施の形態＞

判別分析を効率よく行う手法として、負例を与えることが挙げられる。第１の実施の形態では、正例のみを扱っていたが、第２の実施の形態では、負例をともに扱うことについて説明する。

【０１１７】

（ソフトウェア動作モデル化装置）

第２の実施の形態に係るソフトウェア動作モデル化装置は、図１７に示すように、図１に示すソフトウェア動作モデル化装置に加え、負例分離部１６０を備える。

【０１１８】

負例分離部１６０は、監視対象ソフトウェアの異常動作（例：被攻撃、バグによる動作）が入力された場合、取得したインストラクションから、異常動作によって生じたインストラクションの時系列を分離する。又、負例分離部１６０は、取得したインストラクションを、第１のモデル蓄積部１４０に蓄積された動作モデル上でトレースし、動作モデルと異なるインストラクションが現れた時点からのインストラクション時系列を異常動作によって生じたインストラクション列である負例であると判断する。

【０１１９】

第２のモデル生成部１３２は、負例を用いて、異常動作のモデルを生成する。

【０１２０】

インストラクション取得部１１０、インストラクション蓄積部１２０、第１のモデル生成部１３１、第１のモデル蓄積部１４０、第２のモデル蓄積部１５０については、第１の実施の形態と同様であるので、ここでは、説明を省略する。

【０１２１】

（ソフトウェア動作モデル化方法）

次に、第２の実施の形態に係るソフトウェア動作モデル化方法について、図１８を用いて説明する。

【０１２２】

まず、ステップＳ８０１において、第1のモデル蓄積部１４０よりモデルを、インストラクション蓄積部１１０よりインストラクション時系列をそれぞれ取得する。

【０１２３】

次に、ステップＳ８０２において、取得に成功したか否かを判断し、失敗した場合は、ステップＳ８０３へ進み、エラー処理をして終了する。一方、ステップＳ８０２において、成功した場合は、ステップＳ８０４において、インストラクション時系列からインストラクションを順次読み込む。

【０１２４】

次に、ステップＳ８０５において、読み込みが終了したか否か判断し、終了した場合は、ステップＳ８０６において、エラー処理をして終了する。一方、ステップＳ８０５において、読み込みが終了していなければ、木をトレースする（詳細は、図７参照。）。

【０１２５】

次に、ステップＳ８０８において、木トレースの結果、新規であるか否か判断する。新規である場合は、ステップＳ８０９において、現在位置から終了までの時系列をサブシーケンス化し、このサブシーケンスを負例として抽出し、負例分離を終了する。一方、ステップＳ８０８において、新規でない場合は、ステップＳ８０４へ戻り、次にインストラクションを取得する。

【０１２６】

（作用及び効果）
第２の実施の形態に係るソフトウェア動作モデル化装置及びソフトウェア動作モデル化方法によると、負例を分離することで、異常動作時純粋のデータをモデルにすることができるため、判別分析に効果的な判別空間を生成できる。
【０１２７】
又、木構造モデルを用いることで、インストラクション発行パターンの変化点を正確に検知することができるので、正確な負例抽出が可能である。
【０１２８】
＜第３の実施の形態＞
第１及び第２の実施の形態では、ソフトウェア動作モデル化装置とソフトウェア動作監視装置を異なる装置として説明したが、これらの装置を一つの装置として構成してもよい。
【０１２９】
（ソフトウェア動作モデル化装置）
第３の実施の形態に係るソフトウェア動作モデル化装置１００は、監視対象ソフトウェア１０の動作モデルを生成、蓄積し、蓄積した動作モデルをもとに動作の監視を行う。
【０１３０】
ソフトウェア動作監視装置２００は、監視対象ソフトウェア１０が動作中に発行するインストラクションを取得するインストラクション取得部２１０と、取得したインストラクションの時系列を蓄積するインストラクション蓄積部２２０と、インストラクション蓄積部に蓄積されたインストラクション時系列を学習系列とみなし、動作モデルを生成するモデル生成部１３０と、モデル生成部１３０によって生成された動作モデルを蓄積するモデル蓄積部２８０と、モデル蓄積部２８０に蓄積された動作モデルを用いて監視対象ソフトウェアの正常動作との乖離を判定する動作監視部２９０とを備える。
【０１３１】
インストラクション取得部２１０は、LinuxのPTRACEを利用するなどして、監視対象ソフトウェア１０がインストラクションを発行するたびにインストラクションを取得する。
【０１３２】
モデル生成部１３０は、インストラクション蓄積部２２０に蓄積されたインストラクション時系列を学習系列とし、学習系列をもとに木構造モデルを生成する第１のモデル生成部１３１と、学習系列をもとにインストラクションの共起頻度を計算し、その統計量を特徴量とする動作モデルを生成する第２のモデル生成部１３２とを備える。
【０１３３】
第1のモデル生成部１３１は、例えば、図４に示すフローチャートで動作し、図３に示す木構造データを出力する。第２のモデル生成部１３２は、例えば、図１１に示すフローチャートで動作し、図１４に示すN-gramを出力する。
【０１３４】
モデル蓄積部２８０は、第１のモデル生成部１３１が生成した木構造モデルを蓄積する第１のモデル蓄積部２５０と、第２のモデル生成部１３２が生成したインストラクション発行の統計量（例えば共起頻度）を特徴量とする動作モデルを蓄積する第２のモデル蓄積部２６０とを備える。
【０１３５】
動作監視部２９０は、インストラクション取得部２１０がインストラクションを取得するたびに第１のモデル蓄積部２５０から取得した木構造モデルで監視対象ソフトウェアの発行するインストラクションを検証する第１の検証部２３０と、インストラクション蓄積部２２０に蓄積したインストラクション時系列のうち、第１の検証部２３０が動作の異常を検知してからのインストラクションを検証系列としてとらえ、第２のモデル蓄積部２６０から取得したインストラクション発行の統計量（例えば、インストラクションの共起頻度）を特徴とした動作モデルで監視対象ソフトウェア１０の発行するインストラクションを検証する第２の検証部２４０とを備え、第２の検証部２４０は、第１の検証部２３０が

ソフトウェア動作に異常を検知した場合にのみ動作する。

【０１３６】

第１の検証部２３０は、例えば、図１５に示すフローチャートで動作する。一方、第２の検証部２４０は、例えば、図１６に示すフローチャートで動作する。

【０１３７】

（作用及び効果）

第３の実施の形態に係るソフトウェア動作モデル監視装置によると、自らモデル化し、自ら検証する端末を開発することができる。同一の環境でモデル化、検証を行うことができるため、環境情報を伝えるメタ情報を付与する必要がなく、効率がよい。

【図面の簡単な説明】

【０１３８】

【図１】第１の実施の形態に係るソフトウェア動作モデル化装置の構成ブロック図である。

【図２】第１の実施の形態に係るソフトウェア動作監視装置の構成ブロック図である。

【図３】第１の実施の形態において用いられる木構造の一例である。

【図４】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その１）。

【図５】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その２）。

【図６】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その３）。

【図７】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その４）。

【図８】第１の実施の形態において、蓄積されたインストラクション時系列の一例である。

【図９】第１の実施の形態において、動作モデルの一例である（その１）。

【図１０】第１の実施の形態において、動作モデルの一例である（その２）。

【図１１】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その５）。

【図１２】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その６）。

【図１３】第１の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである（その７）。

【図１４】第１の実施の形態において、N-gramの一例である。

【図１５】第１の実施の形態に係るソフトウェア動作監視方法を示すフローチャートである（その１）。

【図１６】第１の実施の形態に係るソフトウェア動作監視方法を示すフローチャートである（その２）。

【図１７】第２の実施の形態に係るソフトウェア動作モデル化装置の構成ブロック図である。

【図１８】第２の実施の形態に係るソフトウェア動作モデル化方法を示すフローチャートである。

【図１９】第３の実施の形態に係るソフトウェア動作監視装置の構成ブロック図である。

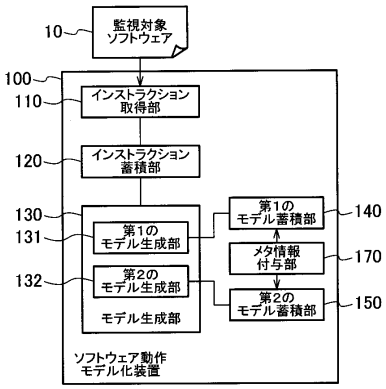
【符号の説明】

【０１３９】

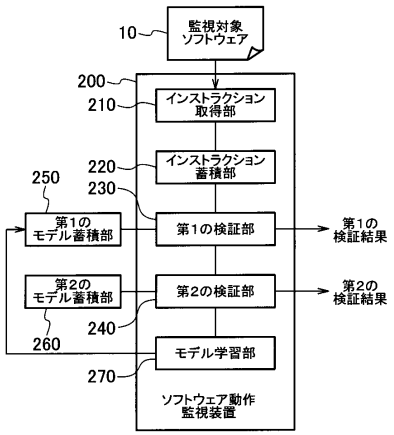
１０…監視対象ソフトウェア
３０…インストラクションテーブル
３１…インストラクション名列
３２…分岐ポインタ列
４０…分岐先アドレステーブル

４１…分岐先アドレス列
１００…ソフトウェア動作モデル化装置
１１０…インストラクション取得部
１１０…インストラクション蓄積部
１２０…インストラクション蓄積部
１３０…モデル生成部
１３１…第1のモデル生成部
１３２…第２のモデル生成部
１４０…第1のモデル蓄積部
１５０…第２のモデル蓄積部
１６０…負例分離部
１７０…メタ情報付与部
２００…ソフトウェア動作監視装置
２１０…インストラクション取得部
２２０…インストラクション蓄積部
２３０…第１の検証部
２４０…第２の検証部
２５０…第１のモデル蓄積部
２６０…第２のモデル蓄積部
２７０…モデル学習部
２８０…モデル蓄積部
２９０…動作監視部

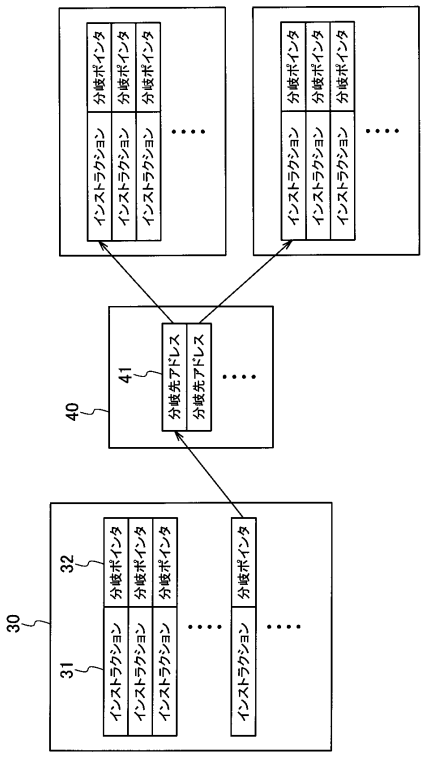
【図１】



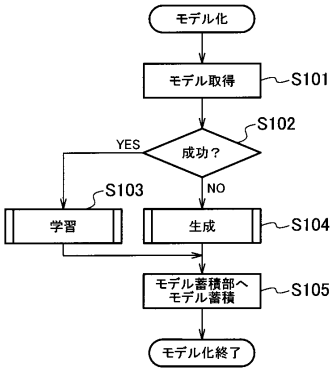
【図２】



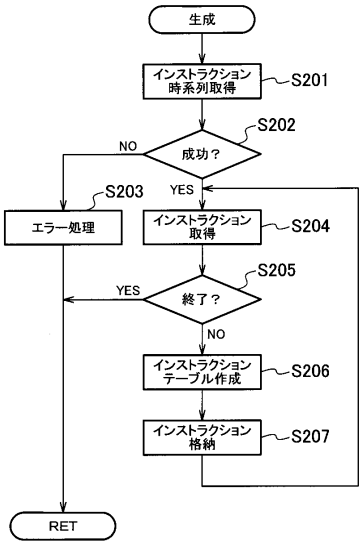
【図３】



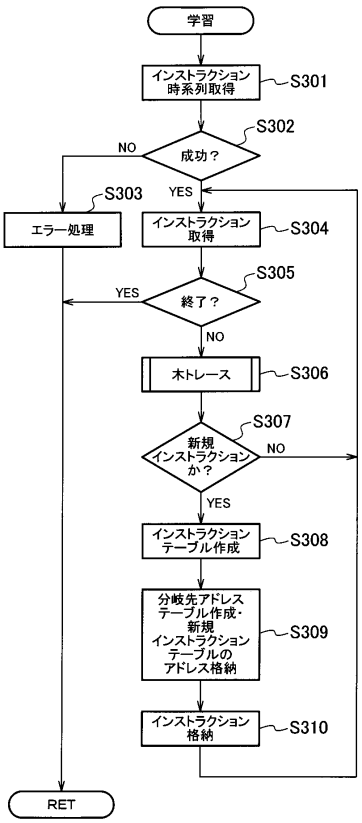
【図４】


モデル化
モデル取得
S101
成功？
S102
YES
NO
学習
S103
生成
S104
モデル蓄積部へモデル蓄積
S105
モデル化終了

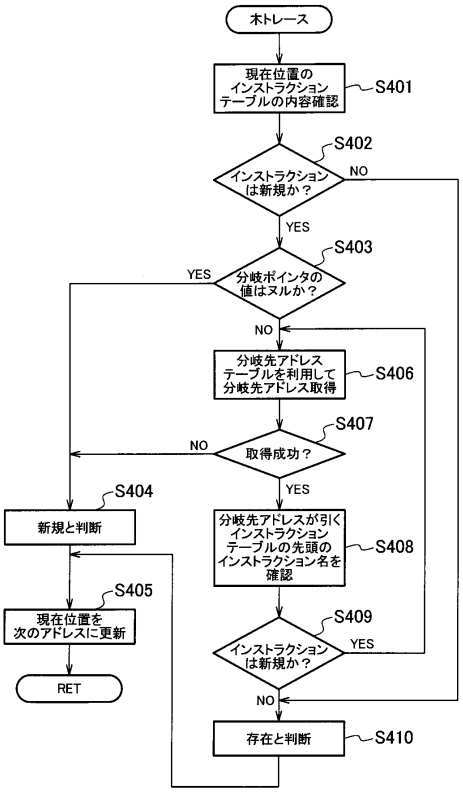
【図５】


生成
インストラクション時系列取得
S201
成功？
S202
NO
YES
エラー処理
S203
インストラクション取得
S204
終了？
S205
YES
NO
インストラクションテーブル作成
S206
インストラクション格納
S207
RET

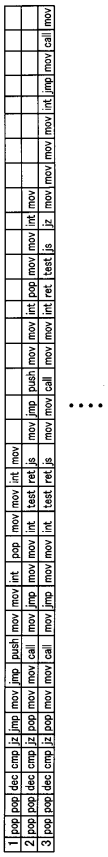
【図６】


学習
インストラクション時系列取得
S301
成功？
S302
NO
YES
エラー処理
S303
インストラクション取得
S304
終了？
S305
YES
NO
木トレース
S306
新規インストラクションか？
S307
NO
YES
インストラクションテーブル作成
S308
分岐先アドレステーブル作成・新規インストラクションテーブルのアドレス格納
S309
インストラクション格納
S310
RET

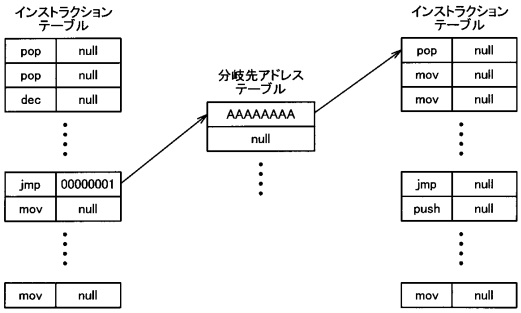
【図７】


木トレース
現在位置のインストラクションテーブルの内容確認
S401
インストラクションは新規か？
S402
NO
YES
分岐ポインタの値はヌルか？
S403
YES
NO
分岐先アドレステーブルを利用して分岐先アドレス取得
S406
取得成功？
S407
NO
YES
新規と判断
S404
分岐先アドレスが引くインストラクションテーブルの先頭のインストラクション名を確認
S408
現在位置を次のアドレスに更新
S405
インストラクションは新規か？
S409
YES
RET
NO
存在と判断
S410

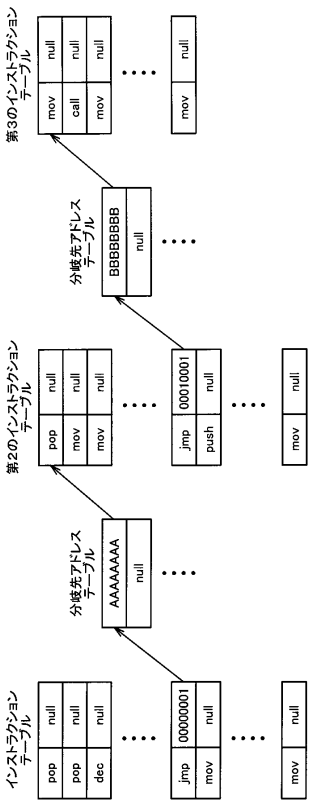
【図８】



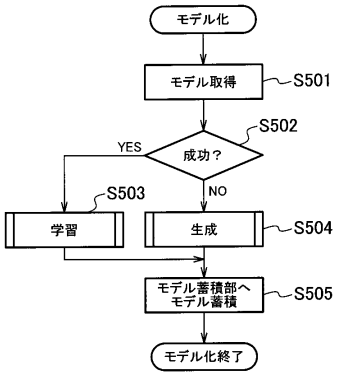
【図９】



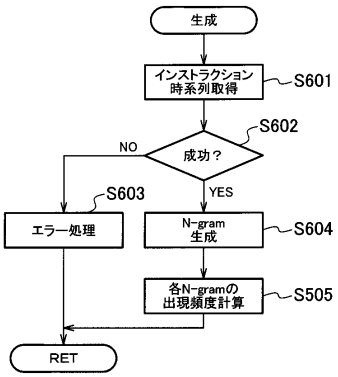
【図１０】



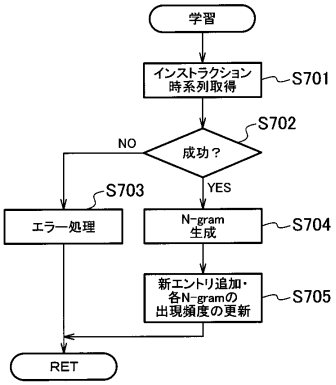
【図１１】



【図１２】



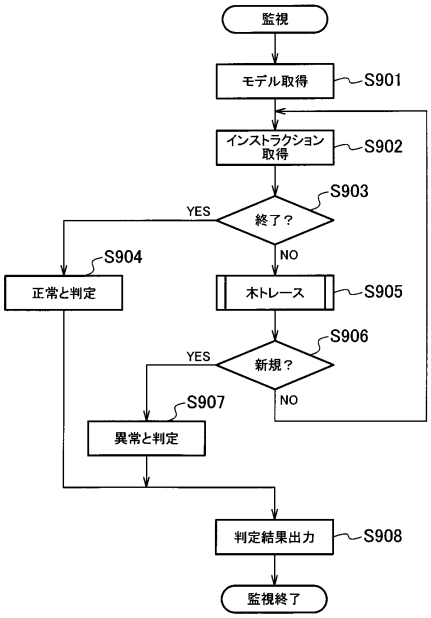
【図１３】



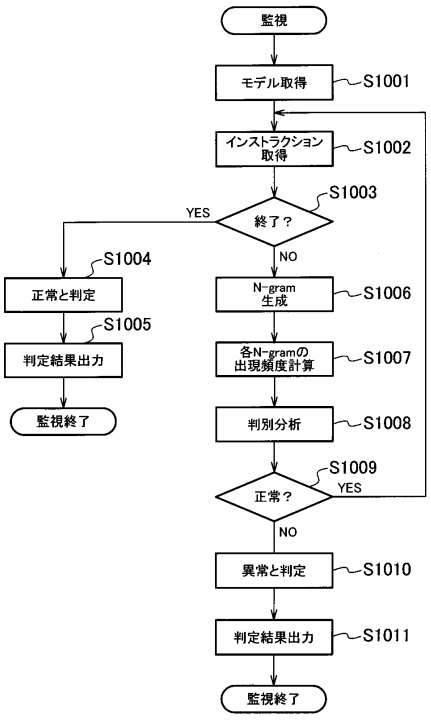
【図１４】

```
xorl xorl xorl ,24
xorl xorl movl ,1964
xorl xorl cmpl ,2
xorl pop pop ,57
xorl pop movl ,1
xorl pop ret ,1457
xorl movl xorl ,5870
xorl movl pop ,6
xorl movl movl ,4334
xorl movl push ,447
xorl movl call ,745
xorl movl jmp ,534
```

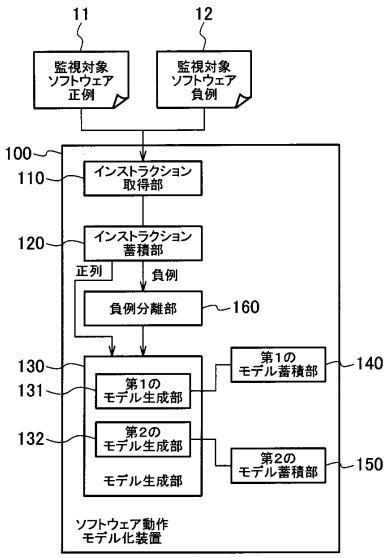
【図１５】



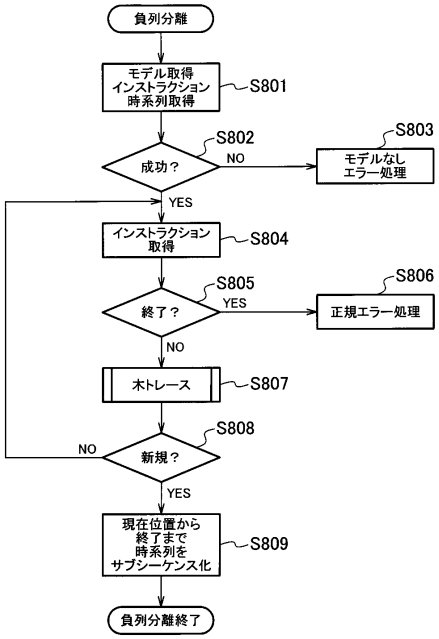
【図１６】



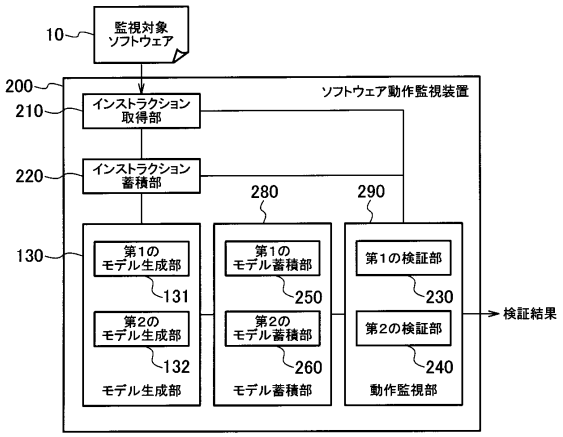
【図１７】



【図１８】


負列分離
モデル取得
インストラクション
時系列取得
S801
S802
成功？
NO
S803
モデルなし
エラー処理
YES
インストラクション
取得
S804
S805
終了？
YES
S806
正規エラー処理
NO
木トレース
S807
S808
新規？
NO
YES
現在位置から
終了まで
時系列を
サブシーケンス化
S809
負列分離終了

【図１９】


10
監視対象
ソフトウェア
200
ソフトウェア動作監視装置
210
インストラクション
取得部
220
インストラクション
蓄積部
280
290
130
第1の
モデル生成部
131
第2の
モデル生成部
132
モデル生成部
第1の
モデル蓄積部
250
第2の
モデル蓄積部
260
モデル蓄積部
第1の検証部
230
第2の検証部
240
動作監視部
検証結果

フロントページの続き

(72)発明者　金野　晃
東京都千代田区永田町二丁目１１番１号　株式会社エヌ・ティ・ティ・ドコモ内
(72)発明者　中山　雄大
東京都千代田区永田町二丁目１１番１号　株式会社エヌ・ティ・ティ・ドコモ内
(72)発明者　竹下　敦
東京都千代田区永田町二丁目１１番１号　株式会社エヌ・ティ・ティ・ドコモ内

審査官　多胡　滋

(56)参考文献　特開２００４－１８５３４５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　２１／００
Ｇ０６Ｆ　１１／２８
Ｇ０６Ｆ　１１／３４
Ｇ０６Ｆ　１１／３６