(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6817469号
(P6817469)
(45)発行日　令和3年1月20日(2021.1.20)　(24)登録日　令和2年12月28日(2020.12.28)

(51)Int.Cl.　　F I
G06F 21/55 (2013.01)　G06F 21/55 320
G06F 11/34 (2006.01)　G06F 11/34 176
G06F 16/903 (2019.01)　G06F 16/903

請求項の数 20　(全 22 頁)

(21)出願番号　特願2019-565938 (P2019-565938)
(86)(22)出願日　平成30年6月13日 (2018.6.13)
(65)公表番号　特表2020-522799 (P2020-522799A)
(43)公表日　令和2年7月30日 (2020.7.30)
(86)国際出願番号　PCT/US2018/037183
(87)国際公開番号　WO2019/032180
(87)国際公開日　平成31年2月14日 (2019.2.14)
審査請求日　令和1年11月28日 (2019.11.28)
(31)優先権主張番号　62/543,032
(32)優先日　平成29年8月9日 (2017.8.9)
(33)優先権主張国・地域又は機関
米国 (US)
(31)優先権主張番号　62/591,819
(32)優先日　平成29年11月29日 (2017.11.29)
(33)優先権主張国・地域又は機関
米国 (US)

(73)特許権者　504080663
エヌイーシー　ラボラトリーズ　アメリカ　インク
ＮＥＣ　Ｌａｂｏｒａｔｏｒｉｅｓ　Ａｍｅｒｉｃａ，　Ｉｎｃ．
アメリカ合衆国　０８５４０　ニュージャージー州　プリンストン　スイート　２００　インディペンデンス　ウェイ　４
(74)代理人　100123788
弁理士　宮崎　昭夫
(74)代理人　100127454
弁理士　緒方　雅昭

最終頁に続く

(54)【発明の名称】コンピュータシステムの脅威検出を改善するためのアプリケーション間依存性分析

(57)【特許請求の範囲】
【請求項１】
コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善するためのシステムであって、
プログラムコードを記憶するためのメモリ装置と、
前記メモリ装置に動作可能に結合され、前記メモリ装置に格納されたプログラムコードを実行することによって前記アプリケーション間依存性分析を実行するように構成されたプロセッサと、を含み、
追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリと、前記追跡分析を実行するための一般的な制約を含む前記ＴＤＬクエリと、分析される前記コンピュータシステムの少なくともイベントを指定する追跡宣言と、前記追跡分析によって生成された追跡グラフを格納する位置を指定する出力仕様と、を受信し、
言語パーサを使用して前記ＴＤＬクエリを構文解析し、
前記追跡分析の結果を生成するために漸進的追跡方法を実施することによって、前記構文解析されたＴＤＬクエリに基づいて前記追跡分析を実行し、
前記ＴＤＬクエリの制約を満たさないすべてのノードを除去し、前記追跡分析の前記結果をクリーニングすることによって追跡グラフを生成し、
前記追跡グラフに基づいて、前記追跡グラフおよび前記追跡グラフの最適化されたバージョンのうちの少なくとも１つを含むクエリ結果を、インタフェースを介して出力する

、システム。
【請求項２】
前記メモリ装置に格納されたプログラムコードは、さらに、
前記追跡分析の前記実行を一時停止し、
更新されたＴＤＬクエリを受信し、
前記更新されたＴＤＬクエリに基づいて前記追跡分析の前記実行を再開する、請求項１に記載のシステム。
【請求項３】
前記メモリ装置に格納されたプログラムコードは、さらに、所与のコンピュータシステムイベントの依存関係を、それぞれが前記依存関係のサブセットを含む複数の実行ウィンドウに分割することによって、前記漸進的追跡方法を実施する、請求項１に記載のシステム。
【請求項４】
実行ウィンドウは３タプル＜begin,finish,e＞として定義され、beginは開始時点であり、finishは終了時点であり、そしてeはコンピュータシステムイベントである、請求項３に記載のシステム。
【請求項５】
前記メモリ装置に格納されたプログラムコードは、さらに、前記複数の実行ウィンドウを、それぞれの終了時点に基づいて優先順位付けすることによって前記漸進的追跡方法を実施する、請求項４に記載のシステム。
【請求項６】
前記メモリ装置に格納されたプログラムコードは、さらに、複数のワーカースレッドを使用して前記漸進的追跡方法を適応的に並列化することによって前記追跡分析を実行する、請求項１に記載のシステム。
【請求項７】
前記漸進的追跡方法は、所与の深さを有するイベントの数が閾値を超えるという決定に応答して、適応的に並列化される、請求項６に記載のシステム。
【請求項８】
前記メモリ装置に格納されたプログラムは、さらに、前記追跡グラフを取り除くことによって前記追跡グラフの最適化されたバージョンを生成し、１または複数のフィルタを使用することによって取り除かれた結果を生成し、前記取り除かれた結果を要約する、請求項１に記載のシステム。
【請求項９】
前記取り除かれた結果は、前記取り除かれた結果からノードを併合することによって要約される、請求項８に記載のシステム。
【請求項１０】
コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善する方法をコンピュータに実行させるためのコンピュータによって実行可能なプログラム命令が具体化されたプログラム命令を有する非一時的なコンピュータ可読記憶媒体を含むコンピュータプログラム製品の前記方法は、
追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリと、前記追跡分析を実行するための一般的な制約を含む前記ＴＤＬクエリと、分析される前記コンピュータシステムの少なくともイベントを指定する追跡宣言と、前記追跡分析によって生成された追跡グラフを格納する位置を指定する出力仕様と、を受信することと、
言語パーサを使用して前記ＴＤＬクエリを構文解析することと、
前記追跡分析の結果を生成するために漸進的追跡方法を実施することを含み、前記構文解析されたＴＤＬクエリに基づいて前記追跡分析を実行することと、
前記ＴＤＬクエリの制約を満たさないすべてのノードを除去することを含み、前記追跡分析の前記結果をクリーニングすることによって追跡グラフを生成することと、

前記追跡グラフに基づいて、前記追跡グラフおよび前記追跡グラフの最適化されたバージョンのうちの少なくとも１つを含むクエリ結果を、インタフェースを介して出力することと、を含む。

【請求項１１】

コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善するためのコンピュータで実施される方法であって、

メモリに動作可能に結合されたプロセッサによって、追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリと、前記追跡分析を実行するための一般的な制約を含む前記ＴＤＬクエリと、分析されるコンピュータシステムの少なくともイベントを指定する追跡宣言と、前記追跡分析によって生成された追跡グラフを格納する位置を指定する出力仕様と、を受信することと、

前記プロセッサによって、言語パーサを使用して前記ＴＤＬクエリを構文解析することと、

前記プロセッサによって、前記追跡分析の結果を生成するために漸進的追跡方法を実施することを含み、前記構文解析されたＴＤＬクエリに基づいて前記追跡分析を実行することと、

前記プロセッサによって、前記ＴＤＬクエリの制約を満たさないすべてのノードを除去することを含み、前記追跡分析の前記結果をクリーニングすることによって追跡グラフを生成することと、

前記プロセッサによって、前記追跡グラフに基づいて、前記追跡グラフおよび前記追跡グラフの最適化されたバージョンのうちの少なくとも１つを含むクエリ結果を、インタフェースを介して出力することと、を含む方法。

【請求項１２】

前記プロセッサによって、前記追跡分析の前記実行を一時停止することと、

前記プロセッサによって、更新されたＴＤＬクエリを受信することと、

前記プロセッサによって、前記更新されたＴＤＬクエリに基づいて前記追跡分析の前記実行を再開することと、をさらに含む、請求項１１に記載のコンピュータで実施される方法。

【請求項１３】

前記漸進的追跡方法を実施する方法は、所与のコンピュータシステムイベントの依存関係を、それぞれが前記依存関係のサブセットを含む複数の実行ウィンドウに分割することをさらに含む、請求項１１に記載のコンピュータで実施される方法。

【請求項１４】

実行ウィンドウは３タプル＜begin, finish, e＞として定義され、beginは開始時点であり、finishは終了時点であり、eはコンピュータシステムイベントである、請求項１３に記載のコンピュータで実施される方法。

【請求項１５】

前記漸進的追跡方法を実施する方法は、前記複数の実行ウィンドウを、それぞれの終了時点に基づいて優先順位付けすることをさらに含む、請求項１４に記載のコンピュータで実施される方法。

【請求項１６】

前記追跡分析を実行することは、複数のワーカースレッドを使用して前記漸進的追跡方法を適応的に並列化することをさらに含む、請求項１１に記載のコンピュータで実施される方法。

【請求項１７】

前記漸進的追跡方法は、所与の深さを有するイベントの数が閾値を超えるという決定に応答して、適応的に並列化される、請求項１６に記載のコンピュータで実施される方法。

【請求項１８】

前記追跡グラフの最適化されたバージョンを生成することをさらに含む、請求項１１に

記載のコンピュータで実施される方法。
【請求項１９】
前記追跡グラフの最適化されたバージョンを生成することは、１または複数のフィルタを使用することによって取り除かれた結果を生成し、前記追跡グラフを取り除くことをさらに含む、請求項１８に記載のコンピュータで実施される方法。
【請求項２０】
前記追跡グラフの最適化されたバージョンを生成することは、前記取り除かれた結果を要約することをさらに含む、請求項１９に記載のコンピュータで実施される方法。
【発明の詳細な説明】
【技術分野】
【０００１】
（関連出願情報）
本出願は２０１７年８月９日に出願された仮出願シリアル番号６２／５４３，０３２、２０１７年１１月２９日に出願された仮出願シリアル番号６２／５９１，８１９、および２０１８年６月１２日に出願された非仮出願シリアル番号１６／００６，１６４の優先権を主張し、これらは全て、その全体が参照により本明細書に組み込まれる。
【０００２】
本発明はデータ処理に関し、より詳細には、コンピュータシステムの脅威検出を改善するためのアプリケーション間依存性分析のためのシステムおよび方法に関する。
【背景技術】
【０００３】
高度持続的脅威（ＡＰＴ）攻撃などのますます巧妙化する攻撃はそれらのステルスおよび複雑さのために、企業情報技術（ＩＴ）セキュリティにとって深刻な課題となっている。ＡＰＴ攻撃は、初期段階、内部調査、横方向移動、最終的にはミッション完了を含む複数の段階で行われる。多くの場合、ＡＰＴ攻撃は、企業ネットワーク内の複数のアプリケーションおよびホストを含むことが多いプロセスによって、企業ネットワークが徐々に侵害される可能性がある。
【発明の概要】
【０００４】
本発明の一態様によれば、コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善するためのシステムが提供される。このシステムは、プログラムコードを記憶するためのメモリ装置を含む。システムはまた、メモリ装置に動作可能に結合されたプロセッサを含む。プロセッサは追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリを受信するようにメモリ装置に格納されたプログラムコードを実行するように構成され、ＴＤＬクエリは追跡分析を実行するための一般的な制約を含み、追跡宣言は分析されるコンピュータシステムの少なくともイベントを指定し、出力仕様は追跡分析によって生成された追跡グラフを格納する位置を指定し、言語パーサを使用してＴＤＬクエリを構文解析し、漸進的追跡方法を実施することによって構文解析されたＴＤＬクエリに基づいて追跡分析を実行し、追跡分析の結果を生成し、ＴＤＬクエリの制約を満たさないすべてのノードを除去し、追跡分析の結果をクリーニングすることによって追跡グラフを生成し、追跡グラフに基いて、追跡グラフおよび追跡グラフの最適化されたバージョンのうちの少なくとも1つを含むクエリ結果を、インタフェースを介して出力する。
【０００５】
本発明の別の態様によれば、コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善するためのコンピュータで実施される方法が提供される。この方法は、メモリに動作可能に結合されたプロセッサによって、追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリと、追跡分析を実行するための一般的な制約を含むＴＤＬクエリと、分析されるコンピュータシステムの少なくともイベントを指定する追跡宣言と、追跡分析によって生成さ

れた追跡グラフを格納する位置を指定する出力仕様と、を受信することと、プロセッサによって、言語パーサを使用してＴＤＬクエリを構文解析することと、プロセッサによって、追跡分析の結果を生成するために漸進的追跡方法を実施することを含み、構文分析されたＴＤＬクエリに基づいて追跡分析を実行することと、プロセッサによって、ＴＤＬクエリの制約を満たさないすべてのノードを除去することを含み、追跡分析の結果をクリーニングすることによって追跡グラフを生成することと、プロセッサによって、追跡グラフに基づいて、追跡グラフおよび追跡グラフの最適化されたバージョンのうちの少なくとも１つを含むクエリ結果を、インタフェースを介して出力することと、を含む。

【０００６】

本発明のさらに別の態様によれば、コンピュータプログラム製品が提供される。コンピュータプログラム製品は、プログラム命令によって具体化されたプログラム命令を有する非一時的なコンピュータ可読記憶媒体を含む。プログラム命令はコンピュータによって実行可能であり、コンピュータに、コンピュータシステムのイベントに対してアプリケーション間依存性分析を実行することによって、コンピュータシステムにおける脅威検出を改善する方法を実行させる。この方法は、追跡分析を実行するための追跡記述言語（ＴＤＬ）クエリと、追跡分析を実行するための一般的な制約を含むＴＤＬクエリと、分析されるコンピュータシステムの少なくともイベントを指定する追跡宣言と、追跡分析によって生成された追跡グラフを格納する位置を指定する出力仕様と、を受信することと、言語パーサを使用してＴＤＬクエリを構文解析することと、追跡分析の結果を生成するために漸進的追跡方法を実施することを含み、構文解析されたＴＤＬクエリに基づいて追跡分析を実行することと、ＴＤＬクエリの制約を満たさないすべてのノードを除去することを含み、追跡分析の結果をクリーニングすることによって追跡グラフを生成することと、追跡グラフに基づいて、追跡グラフおよび追跡グラフの最適化されたバージョンのうちの少なくとも１つを含むクエリ結果を、インタフェースを介して出力することと、を含む。

【０００７】

これらおよび他の特徴および利点は添付の図面に関連して読まれる、その例示的な実施形態の以下の詳細な説明から明らかになるであろう。

【図面の簡単な説明】

【０００８】

本開示は、以下の図面を参照して、好ましい実施形態の以下の説明において詳細を提供する。

【図１】本発明の一実施形態による、本発明を適用することができる例示的な処理システム１００を示すブロック図である。

【図２】本発明の一実施形態による、攻撃例の例示的な依存性グラフを示すブロック図である。

【図３】本発明の一実施形態による、例示的なシステムアーキテクチャを示す高レベルブロック図である。

【図４】本発明の一実施形態による、例示的なＴＤＬクエリを示す図である。

【図５】本発明の一実施形態による、実行ウィンドウの例示的な分割を示す図である。

【図６】本発明の一実施形態による、アプリケーション間依存性分析のための例示的なシステム／方法を示すブロック／フロー図である。

【発明を実施するための形態】

【０００９】

本明細書で説明する実施形態による追跡分析は、異なるアプリケーションとホストとの間のデータフロー（例えば、アプリケーション間データフローまたは情報フロー）に関連するシステムイベント（例えば、システムアクティビティを記録しているシステムアクティビティログ）を監視および追跡することによって、複数のプロセスまたはアプリケーション（例えば、ＡＰＴ攻撃）を含む巧妙なシステムレベルセキュリティ脅威に対して検出し、防御するために使用される。システムイベントはシステムオブジェクト（例えば、プロセス、ファイル、およびネットワーク通信インスタンス）間の対話である。システムイ

ベントは（１）サブジェクト（例えば、対話を開始するプロセスインスタンス）、（２）サブジェクトが対話するシステムオブジェクト、（３）データフローの方向（サブジェクトからシステムオブジェクトへの、またはその逆の）、および（４）対話のタイムスタンプの４つの属性を含むことができる。

【００１０】

本明細書で説明される実施形態による、複数のセキュリティアラートを接続し、攻撃シナリオを再構築するために使用され得る１つの技術は、後戻り追跡である。後戻り追跡技術は、システムイベント間の後方イベント依存性を追跡することができる。事象Ａは（１）Ａの前にＢが発生し、Ｂのデータフローの宛先がＡのデータフローの源である場合、別の事象Ｂに後方依存するといわれる。後方追跡技術は、それらの依存性に基づいてシステム事象を接続する追跡グラフを生成することができる。例えば、システムイベントは有向グラフとして編成することができ、グラフのノードはシステムオブジェクトであり、エッジは開始タイムスタンプを有するシステムイベントであり、追跡分析は有向グラフを検索することによってデータフローを回復することができる。異常が検出された場合、異常の根本原因を回復し、脅威があるかどうかを判定するために追跡分析が使用され得る。

【００１１】

脅威を検出するために追跡分析を実行することに関連する様々な問題が生じる可能性があり、それによって、例えば企業環境において脅威を検出するための追跡分析の有用性が制約される。例えば、ＡＰＴ攻撃のような脅威を検出することは、いくつかの追跡分析の反復を含むことができる。各反復が実行するのに数時間（または数日）を要する場合、脅威は企業環境に既にかなりの損害を引き起こした後に検出され得る。したがって、追跡分析速度は複雑な攻撃（例えば、数時間または数日）に対して法外に長くなり得る。他の例として、追跡分析の結果は何十万ものアイテムを含むことができ、アイテムの多くは、セキュリティ脅威とは無関係である。したがって、追跡分析技術は多くの雑音を含む追跡グラフを生成することができ、これは、システム攻撃を発見するための追跡分析の結果の効果的な解釈を妨げることができる。したがって、追跡分析の結果は、システム攻撃のデータフローが非常に複雑になり得るので、解釈することが困難であり得る。

【００１２】

システム攻撃を検出するための追跡分析に関連する上記の問題（例えば、追跡分析速度および結果解釈の困難性）は、現代のオペレーティングシステムおよび企業環境の複雑さによって引き起こされる。例えば、企業環境では、攻撃に関連するデータフローが多くのアプリケーションおよびホストを含み得る。さらに、現代のオペレーティングシステムはアプリケーション全体にわたる多くのノイズデータフローを生成することができ、攻撃の足跡は、大量のシステムノイズの背後に隠すことができる。例えば、ユーザがフォルダを開くと、フォルダ内の各ファイルからデータフローパスを作成することができる（例えば、フォルダが１００００のファイルを含む場合、１００００のデータフローパスが作成される）。このようなデータフローパスは追跡分析中に追跡することもでき、これは、非常に遅いプロセスをもたらし、解釈不能な結果を生成する可能性がある。さらに、これらのデータフローパスは、セキュリティ脅威とは無関係であるため、攻撃を検出し防御する状況ではノイズである。

【００１３】

システムノイズを除去することは、追跡分析時間を著しく短縮し、はるかに解釈可能な結果を生成することができる。しかしながら、システムノイズを除去する（例えば、フィルタリングする）ための厳密に自動化された技術の使用は不利益に直面する。例えば、攻撃者は、そのような厳密に自動化された技術の利益を利用する新しい攻撃を設計することができる。さらに、追跡分析技術はシステムアクティビティに関する意味レベルの情報を欠いている可能性があり、これは、効果的に追跡分析に自動的に統合することができない。例えば、多くの場合、追跡はライブラリファイル（例えば、.dllファイル）に到達し得る。これらのファイルは、多くの場合、複数のアプリケーションによって共有され、したがって、攻撃に関係しないデータフローを導入する可能性がある。しかしながら、自動化

された技術は、全ての.dllが危険にさらされる可能性があるため、追跡分析から全ての.dllファイルを直接除去することはできない。.dllを除去する前に、ファイルの追跡分析は、未だにマニュアル検査技術を利用して、.dllファイルに不審な変更がないことを確認する必要がある。

【００１４】

コンピュータシステム（例えば、企業環境）上の攻撃（例えば、ＡＰＴ）を検出するための追跡分析技術（例えば、後戻り追跡）に関連する少なくとも上記の懸念および問題に対処するために、本発明の態様は、攻撃検出を改善するためのアプリケーション間データフロークエリシステムおよび方法を提供する。例えば、本明細書で説明される実施形態は悪意のある行動（例えば、ＡＰＴ脅威）を良性の行動（例えば、通常の企業環境動作）から区別する能力を改善するために、知識（例えば、セキュリティの専門知識）を効果的に組み込むことができる。

【００１５】

本明細書に記載の実施形態は、追跡分析をカスタマイズする方法、ならびに追跡分析を「調整可能」にする方法を提供する。すなわち、本明細書で説明される実施形態による攻撃シナリオを再構築するために追跡分析技術を使用することは、「デバッギング」処理と見做すことができ、その結果、追跡分析処理を漸進的に監視することができ、データフローの中間結果を検査することができ、処理対話を統合して追跡分析への案内を提供することができる。したがって、本明細書に記載される実施形態は、セキュリティ専門家が追跡分析を対話的かつ漸進的にデバッグすることを可能にする。

【００１６】

追跡分析をカスタマイズするためのインタフェースを提供するために、本明細書で説明される実施形態は、本明細書で追跡記述言語（ＴＤＬ）と呼ばれるドメイン固有言語（ＤＳＬ）を利用する。ＴＤＬはセキュリティ専門家によって使用され、追跡分析をカスタマイズするために、それらのドメイン知識（例えば、無関係なデータフローを除外するか、または攻撃経路の一部を指定すること）を提供することができる。これらの仕様は、例えば、システムノイズを取り除くために、追跡分析に自動的に組み込まれ得る。例えば、追跡分析（例えば、追跡分析グラフ）の結果から偽陽性を取り除くことができる。

【００１７】

上記の課題にさらに対処するために、本明細書に記載の実施形態は追跡分析の結果を漸進的に報告し、追跡分析を対話型処理に変換することができる。例えば、追跡分析の実行を一時停止することができ、中間結果に基づいてＴＤＬクエリを修正してノイズを除去することができ、修正されたＴＤＬクエリに基づいて追跡分析の実行を再開することができる。

【００１８】

漸進的な追跡分析の滑らかさを確保するために、本明細書で説明される実施形態は追跡分析の各ステップをいくつかの部分に適切に分割し、各部分が終了するのに多くの時間を必要としないことを確保し、結果をリアルタイムまたは略リアルタイムで滑らかに更新することができる。そうすることによって、少なくともいくつかの結果を迅速に得ることができる。ヒントは追跡分析を加速するために発見することができ、追跡分析プロセス全体が終了する前に追跡仕様を最適化することを可能にする。

【００１９】

本明細書に記載される実施形態は追跡分析の速度を加速し、結果の簡潔さを高めることができる。例えば、本明細書に記載される実施形態はマルチスレッド追跡分析を可能にするための改善された並列化スキームを提供することができ、それによって追跡分析の速度を増加させる。したがって、本明細書で説明する実施形態によれば、追跡分析からのより簡潔かつ可読な結果が生成され得るし、それによって攻撃からの回復に関連するコストを低減し得る。

【００２０】

図１は、本発明の一実施形態による、本発明の原理を適用することができる例示的な処

理システム１００を示すブロック図である。処理システム１００は、システムバス１０２を介して他の構成要素に動作可能に結合された少なくとも１つのプロセッサ（ＣＰＵ）１０４を含む。キャッシュ１０６、読出し専用メモリ（ＲＯＭ）１０８、ランダムアクセスメモリ（ＲＡＭ）１１０、入出力（Ｉ／Ｏ）アダプタ１２０、サウンドアダプタ１３０、ネットワークアダプタ１４０、ユーザインタフェースアダプタ１５０、およびディスプレイアダプタ１６０が、システムバス１０２に動作可能に結合される。少なくとも１つのグラフィック処理ユニット（ＧＰＵ）１９４は、システムバス１０２に動作可能に結合される。

【００２１】

第１の記憶装置１２２および第２の記憶装置１２４は、Ｉ／Ｏアダプタ１２０によってシステムバス１０２に動作可能に結合される。記憶装置１２２および１２４は、ディスク記憶装置（例えば、磁気または光ディスク記憶装置）、ソリッドステート磁気装置などの何れであってもよい。記憶装置１２２および１２４は、同じタイプの記憶装置であっても、異なるタイプの記憶装置であってもよい。

【００２２】

スピーカ１３２は、サウンドアダプタ１３０によってシステムバス１０２に動作可能に結合される。トランシーバ１４２は、ネットワークアダプタ１４０によってシステムバス１０２に動作可能に結合される。ディスプレイ装置１６２は、ディスプレイアダプタ１６０によってシステムバス１０２に動作可能に結合される。

【００２３】

第１のユーザ入力装置１５２、第２のユーザ入力装置１５４、および第３のユーザ入力装置１５６は、ユーザインタフェースアダプタ１５０によってシステムバス１０２に動作可能に結合される。ユーザ入力装置１５２、１５４、および１５６は、キーボード、マウス、キーパッド、画像キャプチャ装置、モーションセンシング装置、マイクロフォン、前述の装置のうちの少なくとも２つの機能を組み込んだ装置などの何れかとすることができる。もちろん、本発明の精神を維持しながら、他のタイプの入力装置を使用することもできる。ユーザ入力装置１５２、１５４、および１５６は、同じタイプのユーザ入力装置または異なるタイプのユーザ入力装置とすることができる。ユーザ入力装置１５２、１５４、および１５６は、システム１００との間で情報を入出力するために使用される。

【００２４】

依存性アナライザ１７０は、システムバス１０２に動作可能に結合される。依存性アナライザ１７０は、本明細書で説明される動作のうちの１つまたは複数を実行するように構成される。依存性アナライザ１７０はスタンドアロンの専用ハードウェア装置として実現することができ、またはストレージ装置に格納されたソフトウェアとして実現することができる。依存性アナライザ１７０がソフトウェアで実現される実施形態では、コンピュータシステム１００の別個の構成要素として示されているが、依存性アナライザ１７０は例えば、第１の記憶装置１２２および／または第２の記憶装置１２９に格納され得る。あるいは、依存性アナライザ１７０は別個の記憶装置（図示せず）に格納され得る。

【００２５】

もちろん、処理システム１００は当業者によって容易に考えられるように、他の要素（図示せず）を含んでもよく、また、特定の要素を省略してもよい。例えば、当業者によって容易に理解されるように、様々な他の入力装置および／または出力装置が、処理システム１００の特定の実現に応じて、処理システム１００に含まれ得る。例えば、様々なタイプの無線および／または有線の入力および／または出力装置を使用することができる。さらに、様々な構成の追加のプロセッサ、コントローラ、メモリなども、当業者によって容易に認識されるように利用され得る。処理システム１００のこれらおよび他の変形は、本明細書で提供される本発明の教示を与えられれば、当業者によって容易に考えられる。

【００２６】

さらに、図３に関して以下で説明するアーキテクチャ３００は、本発明のそれぞれの実施形態を実現するためのアーキテクチャであることを認識されたい。処理システム１００

の一部または全部は、アーキテクチャ３００の要素のうちの１つまたは複数において実現することができる。

【００２７】

さらに、処理システム１００は、例えば、図６の方法６００および図７の方法７００の少なくとも一部を含む、本明細書で説明される方法の少なくとも一部を実行することができることを認識されたい。同様に、アーキテクチャ３００の一部または全部を使用して、図６の方法６００および図７の方法７００の少なくとも一部を実行することができる。

【００２８】

図２を参照して、追跡グラフの動機付けの例を説明する。

【００２９】

図２を参照すると、フィッシング電子メール攻撃を示す例示的な追跡グラフ２００が提供されている。この攻撃シナリオでは、攻撃者が不正プログラムを介してファイル（例えば、テキストファイル）をスキャンすることによって、被害者のホストから機密認証情報を盗んでいる。攻撃は、いくつかのステップを有している。第１に、攻撃者は、Ｍｉｃｒｏｓｏｆｔ　Ｅｘｃｅｌ（登録商標）の添付ファイル（「excel.exe」）を含んでいるものとして示される電子メールを被害者に送信する。「excel.exe」ファイルは、不正プログラム（「dropper.exe」）を作成し実行することができる悪意のあるマクロを含み得る。被害者がＭｉｃｒｏｓｏｆｔ　Ｏｕｔｌｏｏｋ（登録商標）（「outlook.exe」）で「excel.exe」を開くと、「dropper.exe」は「cmd.exe」を使用して「findstr.exe」を実行し、ホームディレクトリー上で、被害者のマシン内の重要な認証を検索し、認証を「findstr.out」としてダンプする。それからdropper.exeは、ネットワークを介して攻撃者に「findstr.out」をアップロードする。空間が限られているため、攻撃経路内の重要なオブジェクトのみが図２の追跡グラフに示されている。例えば、＊.dll、＊.iniなどのオブジェクトまたは「findstr.exe」によってスキャンされたファイルの詳細は省略され、追跡グラフ内の１つのノードのみがそれらのセットを表すために使用された。

【００３０】

図２に示す攻撃では「dropper.exe」が実行されると、システム（例えば、企業システム）の異常検出器は未知のプログラムの実行による警告を生成することができる。この警告に基づいて、セキュリティ専門家のチームは攻撃シナリオを再構築し、「dropper.exe」の根本原因を見つけるために後戻り追跡分析を開始することができる。しかしながら、この攻撃の追跡グラフは非常に大きく（例えば、３０．７５Ｋのイベントを含む）、このサイズの追跡グラフを生成するには、多くの時間がかかる可能性がある。このような大きな追跡グラフおよび長い生成時間は、チームが攻撃シナリオの根本原因を効果的に発見することを妨げる可能性がある。

【００３１】

図２の追跡グラフにおけるノードの９９％以上は、無関係なシステムノイズに対応する。ノイズの１つの原因はライブラリ（例えば、.dllファイル）であり、これは、この例示的な例では危険にさらされておらず、攻撃と関係がない。もう１つの原因は、「findstr.exe」であり、色々なアプリケーションで作成された多くの（テキスト）ファイルを読み込み、そのファイルを作成したアプリケーションが、攻撃に無関係の数千のノードに繋がる可能性がある。この攻撃では、その根本的な原因とは対照的に、「findstr.exe」が不正プログラム「dropper.exe」によって使用されるツールである。したがって、この攻撃から「findstr.exe」を除いたものが、根本原因を見つけることから後戻り追跡分析を妨げることはない。

【００３２】

.dllファイルや「findstr.exe」の全てを除去することは、自動化された技術にとって非常に困難であり得る。例えば、.dllファイルを除去するために、後戻り追跡分析は、.dllファイルに不審な変更がないことを最初に確認する必要があり、「findstr.exe」を除去するために、後戻り追跡分析は、「findstr.exe」が根本原因ではなく不正プログラムによって引き起こされたものであることを知る必要がある。しかしながら、この情報を後

戻り追跡分析に自動的に正確に組み込むことは非常に困難である。

【００３３】

攻撃では、セキュリティ専門家がプロセスを「デバッグ」することを可能にする対話型および漸進的な追跡分析（例えば、後戻り追跡分析）を実行することが有用であり得る。図２の追跡グラフの状況を読むことによって、セキュリティ専門家のチームは、本明細書に記載の実施形態によれば、「findstr.exe」が結果であり、「dropper.exe」の原因ではないことを確認することができ、また、後戻り追跡分析から「findstr.exe」も同様に除外することができる。本明細書に記載される実施形態によれば、.dllファイルと「findstr.exe」を除外することによって、図２の追跡グラフのサイズは９９．８％を超えて削減することができ、セキュリティ専門家のチームによって、「dropper.exe」の根本原因を時間ではなく分単位で見つけることができる。

【００３４】

次に、本発明の一実施形態による、コンピュータシステムにおける脅威検出を改善するために対話型および漸進的な追跡分析を適用することができるシステムアーキテクチャについて説明する。

【００３５】

図３は、本発明の一実施形態による、例示的なシステムアーキテクチャ３００を示す高レベルブロック図である。システム３００は、大規模で同種の企業ＩＴ環境に配備されるように設計することができる。

【００３６】

図示のように、システム３００は、言語パーサ３２０、実行部３３０、ホスト３４２およびデータベース（ＤＢ）３４４を有するデータ収集および記憶装置、ならびに結果ビューア３５０を含み得る。

【００３７】

言語パーサ３２０は追跡記述言語（ＴＤＬ）クエリ３１０を受信し、ＴＤＬクエリ３１０を解析するように構成される。例えば、図示のように、言語パーサ３２０は、字句解析構成要素３２２および構文解析構成要素３２４を含むことができる。字句解析構成要素３２２は文字のシーケンスをトークンのシーケンス（例えば、意味が割り当てられた文字列）に変換することを含む語彙分析またはトークン化を実行する。構文解析３２４は入力としてトークンを受け取り、トークンの構造表現を提供するためにデータ構造（例えば、ツリー）を構築する。言語パーサ３２０は別個の字句解析および構文解析３２２および３２４を含むように示されているが、字句解析および構文解析３２２および３２４の機能は単一の構成要素（例えば、スキャナレス構文解析）として具現化され得る。

【００３８】

上述のように、ＴＤＬは、ユーザが追跡分析において条件および制約を簡潔に指定することを可能にするドメイン固有言語である。ＴＤＬクエリ３１０は、例えば、時間範囲、ホスト範囲、後戻り追跡の開始点、後戻り追跡の終了条件、および探索すべき経路を含む制約を指定することができる。ＴＤＬクエリ３１０の実行中に、漸進的に更新される後戻り追跡グラフが出力され得る。

【００３９】

ＴＤＬクエリ３１０は、（１）一般的な制約、（２）追跡宣言、および（３）出力仕様の３つの部分を含むものとして見ることができる。

【００４０】

一般的な制約は、例えば、追跡分析のための時間範囲制約およびホスト範囲制約を含み得る。時間範囲およびホスト範囲は、実際にはシステムログが時間的および特別な局所性を持つ性質を有するので、一般的な制約とみなされる。警報を受信すると、長い履歴を持つすべてのホストのシステムログを探索することが実用的になる。しかし、より現実的な方法は、時間およびホスト範囲の簡潔な表現をサポートする一般的な制約を提供することによって、最近の時間範囲で関連するホストを最初に研究することである。

【００４１】

追跡宣言は、どのイベントを分析すべきか、および追跡分析をいつ終了すべきかを指定する。追跡宣言は、「追跡」陳述および「何処」陳述を含み得る。

【００４２】

追跡陳述は、開始点、終了点、および任意の重要な中間点を含む点を指定する。多くの場合、特定のパターンのみを満たす経路を探索するために後戻り追跡を指定する必要があるため、中間点がサポートされる。追跡陳述は、例示的に以下のように宣言することができる：

```
backward (type var [condition _ list]) (->
  type var [condition _ list]) +
```

【００４３】

この追跡陳述において、追跡陳述の開始を示すキーワード「backward」の後にノードのリストが続く。ノードはイベントのフィルタであり、「type var [condition _ list]」と宣言することができる。システムオブジェクトのタイプを宣言する語句「type」は、プロセスオブジェクトの「proc」、「ファイルオブジェクトの「file」、およびネットワーク接続オブジェクトの「ip」を含む値を有することができる。語句「var」は、ユーザ定義の変数名であり、語句「condition _ list」は、システムオブジェクトをフィルタリングする制約のリストである。制約は、論理演算によって接続され得る。「condition _ list」内の制約は「field op value」の形式において二項演算陳述とすることができ、「field」は変数の属性名である。「field」のオプションのタイプは、共有オプションおよびオブジェクト固有オプション（例えば、「file」オプション、「proc」オプションおよび「ip」オプション）を含む。「event _ id」および「event _ time」などの共有オプションは、すべてのタイプのシステムオブジェクト（例えば、「proc」、「file」および「ip」）で使用することができ、その手段は、ノードが指定されたＩＤおよび時間をそれぞれ有するイベントのみを含むべきである。「file」の場合、可能なオプションは、「filename」、「host」、「path」、「last _ modification _ time」、「last _ access time」、「creation _ time」を含む。「proc」の場合、可能なオプションは、「host」、「exe name」、「pid」、および「starttime」を含む。「ip」の場合、可能なオプションは、「source _ ip」、「destination _ ip」、および「start _ time」を含む。操作「op」は、「<」、「<=」、「>」、「>=」、「=」、および「!=」を含む可能なオプションを有する二項演算である。「op」の後の「value」は、文字列、数値、または時間文字列とすることができる。「value」が文字列である場合、「=」および「!=」は、それぞれ正規表現（regex）の一致および不一致として解釈することができる。

【００４４】

例えば、ノードのリストが$n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_k$のフォーマットを有するように、ｋ個のノードがあると仮定する。このリストでは、n_1は開始点であり、n_kは終了点であり、$n_2 \sim n_{k-1}$は中間点である。「＊」のような記号を終了点として使用して、終了点に関する特定の制約がないことを指定することができる。

【００４５】

where陳述はオプションであり、特定の条件を満たすイベントを除外する制約、または後戻り追跡の時間を制限する制約のような特定のシステムオブジェクトに関連付けられていない制約を定義する。これらの制約は、追跡分析中にシステムオブジェクトをフィルタリングするために使用することができる。where陳述内の制約を満たさないシステムオブジェクトは、さらに探索せずに追跡分析から削除される。where陳述は、以下のように例示的に宣言することができる:

```
where (type. field |hop| time) op value
```

【００４６】

このwhere陳述において、ユーザは、「type. field op value」の形式で制約のリストを指定することができる。制約はまた、論理演算によっても接続される。「type」、「field」、および「op」は、追跡陳述において同じ値のセットを有する。「type field」の他に、where陳述はまた、２つの特別なフィールド、すなわち、「time」および「hop」を

受け入れる。これらのフィールドは追跡分析を終了するために使用することができ、「<=」操作と共に使用することができる。「time」フィールドは追跡分析の時間を制限するために使用することができ、「hop」フィールドは追跡分析における経路の最大長を制限するために使用することができる。追跡分析は「hop」フィールドによって指定された閾値よりも長い経路を見つけると、経路の探索を停止し、もしあれば、他のより短い経路に切り替えることができる。

【００４７】

出力仕様は、生成された追跡グラフをどこに記憶すべきかを指定する。

【００４８】

機密ファイルを盗み、それをネットワークに送る２つの悪意のあるアプリケーションの経路を追跡する例示的なＴＤＬクエリ４００が、図４を参照して示される。図示のように、ＴＤＬクエリ４００の１～２行は、一般的な制約を含む。時間範囲制約はＴＤＬクエリ４００の第１行に見られるように、「from」および「to」というキーワードによって指定することができる。ホスト範囲制約はＴＤＬクエリ４００の２行目に見られるように、キーワード「in」によって指定することができる。この例示的な例では、ＴＤＬクエリ４００は、追跡分析が「０１／０２／２０１７」と「０２／０１／２０１７」の日付の間の「desktop1」および「desktop2」内のシステムイベントの追跡のみを追跡分析することを示す。ＴＤＬクエリ４００の一般的な制約は、オプションの制約であることに留意されたい。一般的な制約が指定されていない場合、すべてのホストがデフォルトの時間範囲で検索される。

【００４９】

図示のように、ＴＤＬクエリ４００の３～７行は、追跡宣言を含む。追跡陳述は、ＴＤＬクエリ４００の３行目における「backward」で開始する。実行中、システム（例えば、図３のシステム３００）は、他の経路の前に中間点を通る経路を自動的に探索する。ＴＤＬクエリ４００の３行目は、「C://Sensitive/important.doc」のファイルに、追跡分析の開始点として「01/16/2017:06:15:14」に書き込むイベントを定義する。ＴＤＬクエリ４００の４行目は、追跡分析の開始点から終了点までの経路がすべて、名前が「malware1」または「malware2」であり、ＩＤが１２であるプロセスを通過すべきであることを示す。ＴＤＬクエリ４００の５行目は、追跡分析の終了点を、ＩＰアドレス「１６８．１２０．１１．１１８」を有するネットワーク通信として定義する。あるいは、終了点に対する制約がない場合、記号（例えば、「＊」）を終了点として定義することができる。

【００５０】

ＴＤＬクエリ４００の６行目の「where」で始まるこの陳述は、追跡分析が実行可能名「explorer」を有する処理を除外すべきであることを示し、実行が１０分を超える場合、プロセス全体が終了されるべきであり、追跡グラフの直径（「hop」）は２５未満であるべきである。

【００５１】

ＴＤＬクエリ４００の８行目の出力仕様は、生成された追跡グラフがパス「./result.dot」に格納されるべきであることを指定する。

【００５２】

図４のＴＤＬクエリ４００などのＴＤＬクエリはシステム（例えば、図３のシステム３００）によって実行される対話型プロセスを反映する複数のバージョンを有することができる。例えば、作成されたＴＤＬクエリ４００の最初のバージョンｖ１は、４行目または７行目なしで作成された。第１のバージョンは、制限時間内に興味深いまたは有意義な結果を発見することができなかったことが決定された。しかし、システムとの対話を通じて情報を得ることによって、追跡分析を加速するための４行目を含むように第２のバージョンｖ２が作成された。同様に、第３のバージョンｖ３は、追跡分析からexplorerを除去するための７行目を含むように作成され、それによって、不審なＩＰを時間制限内に見つけられることを可能にした。

【００５３】

再び図３を参照すると、実行部３３０は言語パーサ３２０によって出力され構文解析されたＴＤＬクエリ３１０に基づいて追跡分析を実行し、追跡分析の結果として追跡グラフを生成する。一般に、実行部３３０は、以下のように動作する。まず、実行部３３０はＤＢ３４４からイベントを検索するためのクエリ（例えば、ＳＱＬクエリ）を生成する。クエリは、結果の更新を漸進的にサポートすることができる。次に、実行部３３０は中間点を追跡する優先順位付け方式を採用することができ、この優先順位付け方式は、指定された中間点を含む経路に優先順位を付けることによって達成することができる。実行部３３０は同じクエリを２回実行することを回避するために、インクリメンタル実行ソリューションを使用することができる。次に、実行部３３０はＴＤＬクエリ３１０によって指定された要件を満たすように最終結果をフィルタリングすることができ、追跡分析処理を加速するために適応的並列化を利用することができる。

【００５４】

より具体的には、実行部３３０が漸進的な追跡分析を実施して追跡分析の結果を漸進的に記録する漸進的実行構成要素３３２を含み得る。これにより、追跡分析全体が完了するのを待つ必要なく、追跡分析の更新された結果を見ることができる。

【００５５】

漸進的実行構成要素３３２によって実施される漸進的な追跡分析は、実例として、実行ウィンドウを分割することによって達成され得る。一般に、システムイベントが多くの他のイベント（「従属」）に依存する場合、漸進的な追跡分析は、依存関係を、各々が従属関係の部分集合を含む複数の実行ウィンドウに分割することができる。追跡分析の結果は、実行ウィンドウの単位で漸進的に更新することができる。イベントが最近であるほど、結果は追跡グラフにより早く更新される。この実行ウィンドウベースのアプローチは、漸進的な追跡分析の滑らかさを改善し得る。例えば、現実世界の環境では、システムイベントが均等に分散されないことがあり、１つのイベントは多数（例えば、数百万）の他のイベントに依存し得る。

【００５６】

形式的には、実行ウィンドウが３－タプル<begin,finish,e>として定義することができ、ここで、beginは開始時点であり、finishは終了時点であり、eは探索される必要があるイベントである。イベントは、実行ウィンドウの単位でＤＢ３４４から取り戻すことができる。

【００５７】

例示的な漸進的な追跡方法を提供する表１が、以下のように提供される。

【００５８】

【表１】

```
Input: e0: start point event
Output: G: backtracking graph
Initialize priQueue ← genExeWindow(e0) and G ← e0;
while priQueue is not empty do
    curr ← priQueue.poll();
    G ← addInComingEdges(curr);
    for Event e in curr.getEdges() do
        priQueue ← genExeWindow(e);
    end
end
return G;
```

【００５９】

表１に示すように、現在のイベントのすべての依存関係をキューに追加する代わりに、

現在のイベントの依存関係を含む実行ウィンドウが１～６行に追加される。グラフ探索のwhileループ（２～６行目）では、漸進的な追跡方法がキューから実行ウィンドウを引き出し、現在の実行ウィンドウ内のすべてのイベントを最終追跡グラフに追加する。イベントは、追跡グラフ（４行目）の端として使用される。次に、ループ（５～７行目）において、漸進的な追跡方法はＤＢ３４４から現在の実行ウィンドウ内で発生するすべてのイベントを列挙し、それらの実行ウィンドウを取得し、これらの実行ウィンドウを将来の探索のためにキューに追加する。関数gen Exe Window（）は、イベントｅを入力として受け入れ、イベントのすべての実行ウィンドウを返す。これを行うため、gen Exe Window（）関数は、入力イベント（te）のタイムスタンプを取得し得る。次に、gen Exe Window（）関数は＜ts,te,e＞として実行ウィンドウを生成することができ、ここでtsは（予め定義された）グローバル開始時間である。一実施形態では、実行ウィンドウはモノリシック実行ウィンドウである。そして、gen Exe Window（）関数は、モノリシック実行ウィンドウをteからtsまでk個に切ることができる。第一実施形態ではkはユーザが構成可能なパラメータである。第１の実行ウィンドウは＜ts_1,te,e＞であり、ここで、$ts_1 = \sigma$および$\sigma = (t_e - t_s) / (2^k - 1)$である。モノリシック実行ウィンドウにさらに時間が残っている場合、第２の実行ウィンドウ＜ts_2,ts_1,e＞が生成され、ここで、$ts_2 = ts_1 - 2\sigma$である。この切断処理は、実行ウィンドウ全体がカバーされるまで繰り返される。したがって、各ステップにおいて、新たに生成された実行ウィンドウの長さは、最後の実行ウィンドウの２倍である。実行ウィンドウの生成を示す例示的なダイヤグラム５００を、図５を参照して説明する。

【００６０】

図５を参照すると、ダイヤグラム５００は６つの実行ウィンドウ（例えば、ｋ＝６）を描くことを示す。入力イベントは、ダイヤグラム５００において端としてマークされている。仕切りは、時間の降順で左から右に始まる。図示されるように、２つの連続する実行ウィンドウの対は、前者の２倍の長さのウィンドウサイズを有する。

【００６１】

再び図３を参照すると、表１に示すように、生成された実行ウィンドウは、１行目および６行目で優先キュー、priキューに追加される。優先キューは、実行ウィンドウの終了時間に基づいて実行ウィンドウに優先する。例えば、ユーザは、より最近のデータを気にするので、より最近の終了時間を有する実行ウィンドウが、列内で優先され、より早い終了時間を有する実行ウィンドウの前に配置される。このように実行ウィンドウに優先順位を付けることによって、より最近の結果がより早く返される可能性が高い。同様の理由で、より小さい実行ウィンドウは、より最近のデータに割り当てられ得る。

【００６２】

追跡分析の速度を加速するために、追跡分析処理を並列化することができる。例えば、図示のように、実行部３３０は、追跡分析処理を並列化するために適応解を利用する適応並列化構成要素３３４を含み得る。言い換えれば、適応並列化構成要素３３４は、長期間続くと予想される追跡分析を並列化するだけである。追跡分析を並列化することは計算オーバヘッドを有するので、適応並列化を実行することができる。さらに、追跡分析が短期間内に終了する場合、並列化は実質的な利点をもたらさない。企業環境では、多くのＴＤＬクエリが存在する可能性がある。短期間の追跡分析に過度に多くの計算資源が費やされるのであれば、複数のＴＤＬクエリを適用することは拡張可能性がない。

【００６３】

より具体的には、現在の追跡分析が潜在的に長時間持続すると判定された場合、適応並列化構成要素３３４は漸進的な追跡方法（例えば、表１）を実行するｎ個のワーカースレッドを使用することができる。現在の追跡分析が潜在的に長時間持続することができるかどうかを判定するために、追跡経路内の深さｄを有する多数のイベントをチェックすることができる。深さｄを有するイベントの数が閾値Ｔを超える場合、現在の追跡分析が潜在的に長時間続き得ると判定される。一実施形態では、ｄ＝３およびＴ＝１０である。ｎ個のワーカースレッドは同じ優先キューを共有することができ、キューから排他的実行ウィ

ンドウを同時に読み込み、データベース３４２からイベントを取り戻し、出力を更新し、新しく生成された実行ウィンドウを優先キューに挿入することができる。

【００６４】

システム３００は、中間点を順次通過する経路のみを探索すべきである。上述したように、システム３００が全てのイベントの探索を終了することなく（例えば、先見性がない場合であってもイベントを取り除くために）各中間ポイントを順次進める方法を知るために、実行部３３０は、優先順位付け方式を活用して中間ポイントを追跡することができる。一般に、中間点のプレフィックスを含む追跡グラフのサブ経路が見つかると、そのサブ経路を他の方向の前に探索することができる。一実施形態では、状態伝播方法を使用して、優先順位付け方式を達成することができる。例えば、ＴＤＬクエリ３１０の追跡宣言声明において宣言されたノードのリストが、フォーマット$n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_k$を有すると仮定する。このリストでは、$n_1$が始点であり、$n_k$が終点であり、$n_2 \sim n_{k-1}$が中間点である。実行部３３０は、各ノードを対応する状態に割り当てることができる。追跡分析中に、状態siに関連付けられた現在のノードn_iがn_{i+1}（ｉ＋１＜ｋ）の制約を満たす後続ノードを有することを実行部３３０が見つけた場合、実行部３３０は、n_{i+1}を状態s_{i+1}に割り当てることができる。

【００６５】

実行部３３０はユーザがＴＤＬクエリ３１０の実行を一時停止して、ユーザがＴＤＬクエリ３１０を更新し、更新されたＴＤＬクエリに基づいて実行を再開することを可能にすることができる。実行部３３０は追跡分析における制約を自動的に更新し、更新されたＴＤＬクエリに基づいて実行を継続することができる。この処理では、実行における増分更新が行われ、前回の実行は最初からやり直されない。したがって、実行部３３０は、更新されたＴＤＬクエリの新しい制約を使用して、将来の探索をガイドする。

【００６６】

増分更新をサポートするために、実行部３３０は、以下の解決策を活用することができる。ある時点で、ユーザが実行を一時停止し、ＴＤＬクエリＣを更新されたバージョンＣ´に更新し、その後、実行を再開すると仮定する。実行部３３０は更新されたＣ´の制約条件を受け取ると、Ｃ´で指定された開始点がＣで指定された開始点と同じであるか否かを判定する。同じでない場合、ユーザが異なる開始点から新たな追跡分析を開始したいことを意味し、実行部３３０は、現在の分析を放棄する。開始点が変更されない場合、ユーザがいくつかの新しい条件を追加しただけであることを意味する。この場合、実行部３３０は、次いで、開始点から現在の探索された追跡グラフを精査し、上述の優先順位付け方式（例えば、状態伝播方法）をやり直すことによって、各ノードの状態を再計算する。このとき、追跡グラフは既にメモリにキャッシュされているので、優先順位付け方式は以前よりも高速に実行され得ることに留意されたい。状態が再計算された後、実行部３３０は検出されるが探索されないノードのセットである追跡分析の「フロンティア」内のノードを再順序付けし、フロンティア内のノードをフィルタリングするために新しい条件を使用する。フロンティアが更新された後、実行部３３０は、新しい制約に基づいて現在の実行を再開することができる。

【００６７】

追跡分析が終了した後、実行部３３０は（更新された）ＴＤＬクエリの制約を満たさないすべてのノードを除去するために、最終追跡グラフをクリーニングすることができる。これを行うために、実行部３３０はすべてのノードを解析し、ＴＤＬクエリのwhereステートメントにおける制約を満たさないノードを除去し、最終追跡グラフ（例えば、メモリ内にキャッシュされている）を精査して、開始点に接続されていないノードを除去し、中間点の制約を満たさないノードおよびエッジを除去することができる。例えば、ＤＦＳ（Depth First Search）を適用して、始点から終点までのすべての経路を再帰的に見つけ、中間点の制約を満たす経路を最終結果に追加することができる。開始点と終了点との間の経路の数は追跡グラフのサイズに対して指数関数的であり得るが、全ての経路がリスト化される必要はない。パスを生成する間、実行部３３０は、それがどの中間点を通過するか

を維持する。したがって、実行部３３０は経路が終点に到達したとき（例えば、Ｏ（１）時間の複雑さで）、経路が中間点の制約を満たすかどうかを迅速に判定することができる。したがって、実行部３３０は、ＤＦＳと同じ時間複雑性で最終追跡グラフをクリーニングすることができる。

【００６８】

結果ビューワ３５０は実行部３３０によって生成された追跡グラフを入力として受け取り、追跡グラフのグラフィカルビューを含むクエリ結果３６０を出力するインタフェース（例えば、グラフィカルユーザインターフェース（ＧＵＩ））を提供する。結果ビューア３５０は追跡分析に関するユーザの見通しを助けるために、追跡グラフのフルバージョンに基づいて追跡グラフの最適化バージョンを作成し得る。追跡グラフの最適化されたバージョンは、追跡グラフのフルバージョン内のいくつかのノードを除去および／または併合することによって作成することができる。結果ビューワ３５０は、追跡グラフから如何なるノードも削除せず、ユーザからいくつかのノードを隠すことに留意されたい。したがって、ユーザは、結果ビューア３５０において、追跡グラフのフルバージョンと追跡グラフの最適化バージョンとの間で切り替えることができる。最適化の複数のタイプを結果ビューワ３５０に組み込むことができる。例えば、図示のように、結果ビューアは、除去構成要素３５２および要約構成要素３５４を含むことができる。

【００６９】

除去構成要素３５２は、実行部３３０によって出力された結果を取り除いて、データフローを他のノードに伝播しない「デッドエンド」ノードを排除する。一実施形態では、除去構成要素３５２が追跡グラフを取り除くために１つまたは複数のフィルタを使用する。例えば、１つ以上のフィルタは、分析されている期間内に読み出されたが書き込まれなかったファイルを排除する読み出し専用フィルタを含むことができる。しばしば、これらのファイルは、一般にセキュリティ攻撃に関係しないデフォルト構成または共通ライブラリである。他の例として、１つまたは複数のフィルタは、１つの処理によってのみ読み書きされたファイルを取り除く自己読み出し専用フィルタを含むことができる。このようなファイルは通常、ログであり、一般に、セキュリティ攻撃とは無関係である。

【００７０】

要約構成要素３５４は結果をより簡潔にするために、取り除かれた結果を要約する。例えば、要約構成要素３５４は、追跡グラフ内のノードを併合することができる。追跡グラフ内のノードを併合するために使用され得る方法は、（１）同様のノードを併合すること、（２）過渡的な処理を併合すること、および（３）同じリモートＩＰに接続されたソケットを併合することを含む。

【００７１】

同様のノードを併合する方法には以下の４つの記述が真である場合、ノードＡおよびＢは類似している：（１）ＡおよびＢはそれぞれ、１つの先行および１つの後続のみを有する；（２）ＡおよびＢの親は同じである；（３）ＡおよびＢの子は同じである；および（４）ＡおよびＢは同じタイプのシステムオブジェクトを表す。これらの類似のノードは、通常、バッチとして生成され、追跡グラフにおいて類似の意味を有する。要約構成要素３５４は、追跡グラフの最適化されたバージョン内の１つのノードとして類似のノードを併合することができる。

【００７２】

併合過渡的処理方法では追跡グラフ内のノードが以下の３つの条件を満たす場合、過渡処理ノードとして定義される：（１）ノードはプロセスを表し；（２）ノードは１つの先行ノードおよび１つの後続ノードのみを有し、先行ノードおよび後続ノードの両方は処理であり；（３）ノードの実行可能名は、その先行ノードの実行可能名と同じである。過渡処理は、その親に併合される。

【００７３】

リモートホストにアクセスするとき、ローカルホストは同じリモートＩＰアドレスおよびポートを有する同じ処理に接続するための、異なるソケットを生成することができる。

同じ遠隔ＩＰ進入路に接続された併合ソケットでは、同じ入出力エッジを有するすべての隣接するソケットノードを単一の仮想ソケットノードに併合することができる。

【００７４】

図２の動機付けとなる例に戻って参照すると、この攻撃事例の攻撃シナリオを再構築するために、チームは、警告の根本原因についての考えを持っていないのであるが、例示的なセキュリティ専門家チームが、「dropper.exe」（例えば、異常検出器によって警告された後）を実行するイベントから後戻り追跡を開始した。したがって、現時点では如何なるガイダンスもなく、チームは基本的な後戻り追跡分析を実行することしかできなかった。そうするために、チームは、初期ＴＤＬクエリとして基本的な後戻り追跡のために以下のＴＤＬクエリを実行した：

```
1 from "01/02/2017" to "02/01/2017"
2 backward proc alert [exename = "dropper.exe"
      and event _ time = "01/17/2017:03:01:07"
      and type="start"]-> *
3 output ="./result.dot"
```

【００７５】

初期ＴＤＬクエリは、与えられた時間に「dropper.exe」という名前の処理を開始するイベントから後戻り追跡が開始すること、および、追跡分析が、１行目の「from to」記述で与えられた開始点のデータ依存関係を１月以内に検索することを宣言した。初期ＴＤＬクエリはまた、追跡グラフを「result.dot」にファイルに格納するために、３行目に出力仕様を提供する。しかし、初期ＴＤＬクエリ内に他の制約は提供されなかった。

【００７６】

初期ＴＤＬクエリが実行されると、追跡グラフが漸進的に表示され始めた。１分以内に２つのイベントを見た後、チームは追跡グラフが「excel.exe」を含むことに気づき、これはたまたま多くの.dllファイルをロードした。この時点で、チームは追跡分析を一時停止し、異常検出器からの他の警報を検索し、.dllファイルに不審な変更がないことを見出した。したがって、チームは、攻撃は、.dllファイルに注入されたコードからのものではなく、追跡分析の焦点は、他のデータ依存関係に置かれるべきであると断定した。したがって、チームは、以下に示すwhere記述を含むことによって、すべての.dllファイルを除外するために初期ＴＤＬクエリを修正した。

```
1 from "01/02/2017" to "02/01/2017"
2 backward proc alert [exename = "dropper.exe"
      and event time = "01/17/2017:03:01:07"
      and type="start"] -> *
3 where file. path != "*.dll"
4 output = "./result. dot"
```

【００７７】

この修正の後、チームは、更新されたＴＤＬクエリを用いて追跡分析の実行を再開した。２分間でさらに８つのイベントを見た後、チームは追跡グラフが「findstr.out」を介して「findstr.exe」に到達したことに気づいた。「findstr.exe」の後の最初の１００のイベントを見た後、チームは「findstr.exe」が多くのファイルをスキャンするために使用されること、および「findstr.exe」の後の依存関係グラフを完全に探索するのに長い時間がかかることに気付いた。チームはさらに、「findstr.exe」がその根本的な原因ではなく、むしろ「dropper.exe」によって使用される可能性が高いことに気づいた。したがって、セキュリティチームは追跡分析を再び一時停止し、以下のようにwhere記述を変更することによって、グラフから「findstr.exe」を除外するように更新されたＴＤＬクエリを修正した。

```
1 from "01/02/2017" to "02/01/2017"
2 backward proc alert [exename = "dropper.exe"
      and event time = "01/17/2017:03 :01 :07"
```

```
        and type="start"] -> *
3 where file.path != "*.dll" and proc.exename
        != "findstr.exe"
4 output = "./result.dot"
```

【００７８】

この修正の後、チームは、新たに更新されたＴＤＬクエリを用いて追跡分析の実行を再開した。約４分後、チームは「outlook.exe」を見つけ、約３０回以上のイベントをチェックすることによって、それに接続されたソケットを見つけた。この時点で、チームは、「outlook.exe」によって生成された「excel.exe」によって「dropper.exe」が生成されたことを発見した。このことから、彼らは「dropper.exe」の根本原因がフィッシング電子メールであることを確認した。この時点までに、チームは追跡分析を行い、合計約１４０のイベントをチェックするのに約７分間を費やした。したがって、本明細書で説明される実施形態に従って実行されるアプリケーション間依存性分析処理は、例えば、セキュリティ脅威または攻撃の根本原因を発見するための時間を低減することによって、コンピュータシステムにおける脅威検出を改善する。

【００７９】

図６は、本発明の一実施形態による、アプリケーション間依存性分析のための例示的な方法６００を示す流れ図である。

【００８０】

ブロック６１０において、追跡分析を実行するためのＴＤＬクエリが受信される。上述のように、ＴＤＬは、ユーザが追跡分析において条件および制約を簡潔に指定することを可能にするドメイン特有の言語である。ＴＤＬクエリ３１０は、例えば、時間範囲、ホスト範囲、後戻り追跡の開始点、後戻り追跡の終了条件、および探索すべき経路を含む制約を指定することができる。ＴＤＬクエリの実行中に、漸進的に更新される後戻り追跡グラフが出力され得る。ＴＤＬクエリは、（１）追跡分析を実行するための一般的な制約；（２）分析されるべきコンピュータシステムの少なくともイベントを指定する追跡宣言；および（３）追跡分析によって生成された追跡グラフを格納するための位置を指定する出力仕様の３つの部分を含んでいるものと見なすことができる。ＴＤＬクエリに関するさらなる詳細は、図３および図４を参照して上述されている。

【００８１】

ブロック６２０において、ＴＤＬクエリは、自然言語パーサを使用して構文解析される。ＴＤＬクエリの構文解析に関するさらなる詳細は、図３を参照して上述されている。

【００８２】

ブロック６３０において、追跡分析は、追跡分析の結果を生成するために漸進的な追跡方法を実施することによって、構文解析されたＴＤＬクエリに基づいて実行される。例えば、１つまたは複数のクエリを実行して、データベースから１つまたは複数のイベントを取り戻すことができ、漸進的な追跡方法は、追跡分析の結果を漸進的に記録して、追跡分析全体が完了するのを待つ必要なく、追跡分析の更新された結果を見ることができるようにすることが可能である。一実施形態では、追跡分析の速度を加速するために（例えば、現在の追跡分析が長時間持続する可能性があると判定された場合）、複数のワーカースレッドを使用して漸進的な追跡方法を実行することによって、追跡分析を適応的に並列化することができる。追跡分析の実行はＴＤＬクエリの更新を可能にするために一時停止することができ、更新されたＴＤＬクエリに基づいて再開することができる。例えば、追跡分析における制約は更新されたＴＤＬクエリに基づいて自動的に更新することができ、更新は、以前の実行が最初からやり直されないようなインクリメンタル更新とすることができる。ブロック６３０に関するさらなる詳細は、図３を参照して上述されている。

【００８３】

ブロック６４０において、追跡分析の結果をクリーニングすることによって追跡グラフが生成される。例えば、追跡グラフは、（更新された）ＴＤＬクエリの制約を満たさない全てのノードを除去することによってクリーニングすることができる。

【００８４】
ブロック６５０において、追跡グラフに基づくクエリ結果がインタフェースを介して出力される。一実施形態では、インタフェースはＧＵＩを含む。追跡グラフの最適化されたバージョンは、追跡グラフのフルバージョンに基づいて、例えば追跡グラフのフルバージョン内のノードのいくつかを除去および／または併合することによって、作成することができる。例えば、ブロック６３０における追跡分析によって出力された結果は１つまたは複数のフィルタ（例えば、読み取り専用フィルタおよび／または自己読み取り専用フィルタ）を使用することによって「dead end」ノードを排除するように取り除くことができ、取り除かれた結果は、クエリ結果をより簡潔にするように要約され得る。ブロック６４０に関するさらなる詳細は、図３を参照して上述されている。

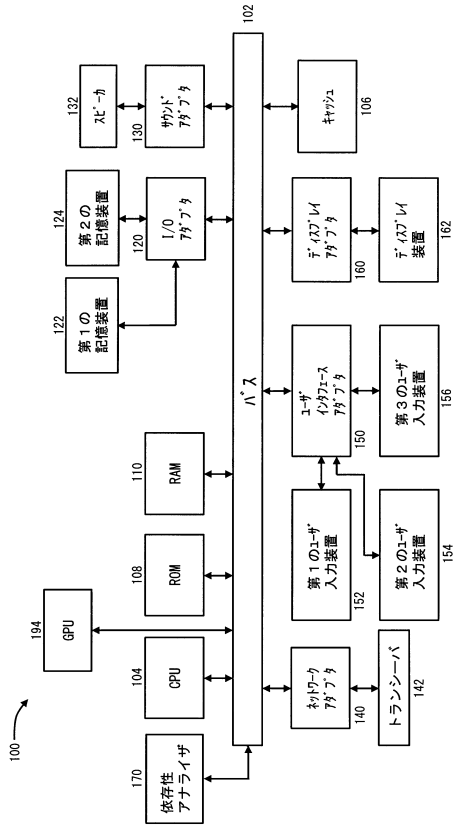
【００８５】
本明細書で説明される実施形態は、完全にハードウェアであってもよく、完全にソフトウェアであってもよく、またはハードウェア要素とソフトウェア要素の両方を含んでもよい。好ましい実施形態では、本発明がファームウェア、常駐ソフトウェア、マイクロコードなどを含むがこれらに限定されないソフトウェアで実現される。

【００８６】
実施形態は、コンピュータまたは任意の命令実行システムによって、またはそれに関連して使用するためのプログラムコードを提供する、コンピュータ使用可能媒体またはコンピュータ可読媒体からアクセス可能なコンピュータプログラム製品を含むことができる。コンピュータ使用可能媒体またはコンピュータ可読媒体は、命令実行システム、装置、またはデバイスによって、またはそれに関連して使用するためのプログラムを格納、通信、伝搬、または運搬する任意の装置を含むことができる。媒体は、磁気、光学、電子、電磁気、赤外線、または半導体システム（または装置またはデバイス）、または伝搬媒体とすることができる。媒体は、半導体またはソリッドステートメモリ、磁気テープ、リムーバブルコンピュータディスケット、ランダムアクセスメモリ（ＲＡＭ）、リードオンリメモリ（ＲＯＭ）、リジッド磁気ディスク、および光ディスクなどのコンピュータ可読媒体を含むことができる。

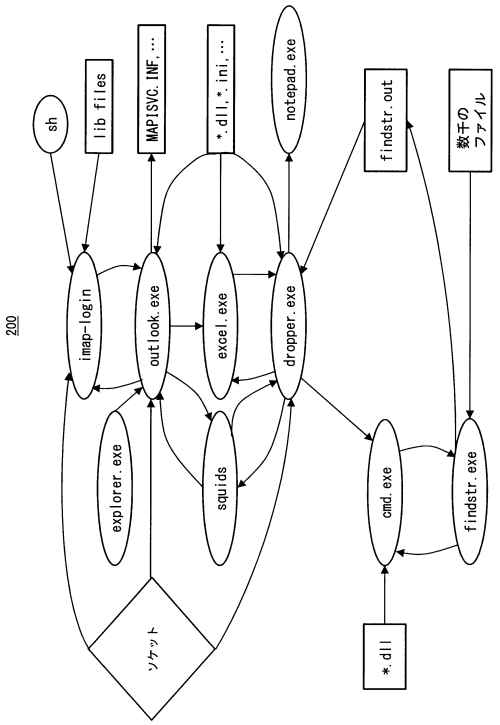
【００８７】
以下の「／」、「および／または」、および「例えば「Ａ／Ｂ」、「ＡおよびＢ」および「ＡおよびＢの少なくとも１つ」の場合の少なくとも１つの使用は、第１のリスト化されたオプション（Ａ）のみの選択、または第２のリスト化されたオプション（Ｂ）のみの選択、または両方のオプション（ＡおよびＢ）の選択を包含することが意図されることを理解されたい。さらなる例として、「Ａ、Ｂ、および／またはＣ」および「Ａ、Ｂ、およびＣのうちの少なくとも１つ」の場合において、このような句は、第１のリスト化されたオプション（Ａ）のみの選択、または第２のリスト化されたオプション（Ｂ）のみの選択、または第３のリスト化されたオプション（Ｃ）のみの選択、または第１および第２のリスト化されたオプション（ＡおよびＢ）のみの選択、または第１および第３のリスト化されたオプション（ＡおよびＣ）のみの選択、または第２および第３のリスト化されたオプション（ＢおよびＣ）のみの選択、または３つすべてのオプション（ＡおよびＢおよびＣ）の選択を包含することが意図される。これは、当業者には容易に明らかなように、列挙された項目の数だけ拡張することができる。

【００８８】
システムおよび方法の好ましい実施形態（これは例示的であり、限定的ではないことが意図される）を説明したが、上記の教示に照らして、当業者によって、修正および変形がなされ得ることに留意されたい。したがって、添付の特許請求の範囲によって概説されるような本発明の範囲および精神内にある、開示された特定の実施形態に変更を加えることができることを理解されたい。このように、本発明の態様を、特許法によって要求される詳細および特殊性と共に説明してきたが、特許証によって保護されることが請求され、望まれるものは添付の特許請求の範囲に記載されている。

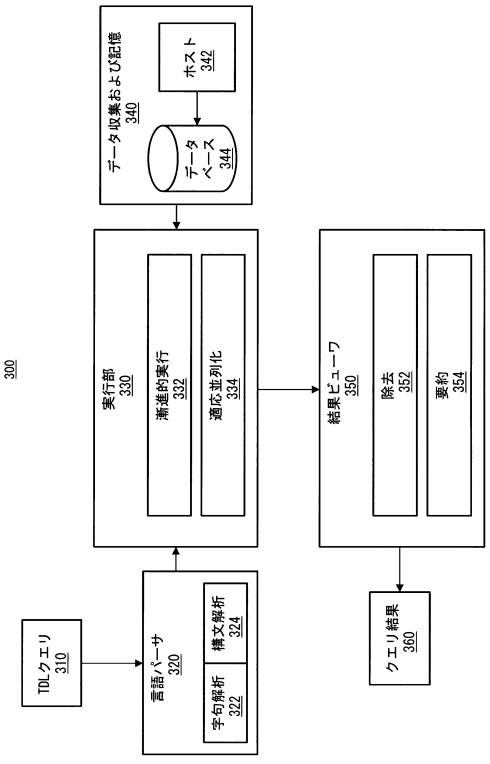
【図１】



【図２】



【図３】



【図４】

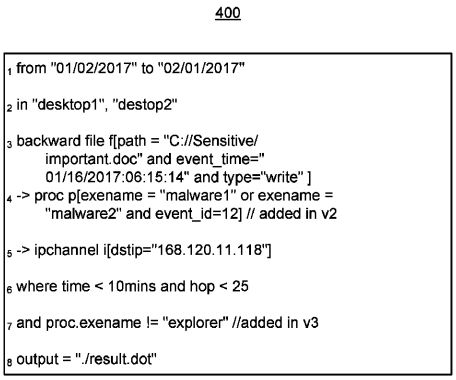
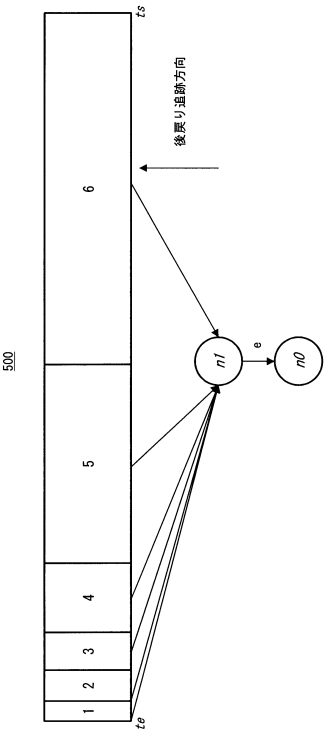
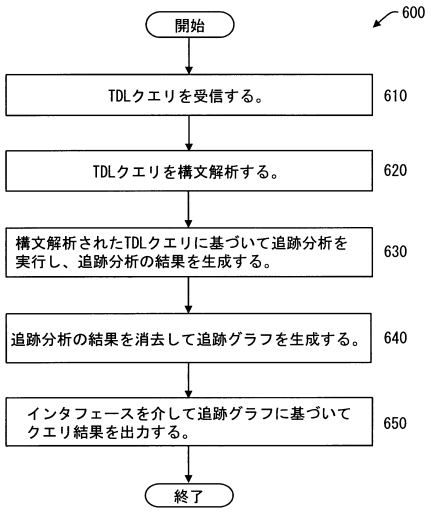


FIG. 4

【図５】


500
ts
te
後戻り追跡方向
6
5
4
3
2
1
n1
e
n0

【図６】


600
開始
TDLクエリを受信する。
610
TDLクエリを構文解析する。
620
構文解析されたTDLクエリに基づいて追跡分析を実行し、追跡分析の結果を生成する。
630
追跡分析の結果を消去して追跡グラフを生成する。
640
インタフェースを介して追跡グラフに基づいてクエリ結果を出力する。
650
終了

フロントページの続き

(31)優先権主張番号 16/006,164
(32)優先日 平成30年6月12日(2018.6.12)
(33)優先権主張国・地域又は機関
米国(US)

(72)発明者 リ、 ディン
アメリカ合衆国 ０８５５０ ニュージャージー州 ウェスト ウィンザー テイラー コート １２２１１
(72)発明者 ジー、 カンクック
アメリカ合衆国 ０８５４０ ニュージャージー州 プリンストン ジョナソン デイトン コート １９２
(72)発明者 チェン、 ジェンジャン
アメリカ合衆国 ０８５５０ ニュージャージー州 プリンストン ジャンクション ヨーク ロード ４４
(72)発明者 タン、 ルーアン
アメリカ合衆国 ０８５３４ ニュージャージー州 ペニントン マンリー ロード １０
(72)発明者 リ、 ジチュン
アメリカ合衆国 ０８５４０ ニュージャージー州 プリンストン セイヤー ドライヴ ３０６ アパート４１３シー

審査官 岸野 徹

(56)参考文献 米国特許出願公開第２０１６／０３０８７２５（ＵＳ，Ａ１）
米国特許出願公開第２０１７／０２２０６３９（ＵＳ，Ａ１）
特開２０１２－２４３１２３（ＪＰ，Ａ）
特開２０１５－１３３０９７（ＪＰ，Ａ）
米国特許出願公開第２０１６／０３５９８７７（ＵＳ，Ａ１）
米国特許出願公開第２０１６／０２９９９８２（ＵＳ，Ａ１）
米国特許出願公開第２０１２／０１１０５９９（ＵＳ，Ａ１）
米国特許出願公開第２０１２／０２９６９２３（ＵＳ，Ａ１）
中国特許出願公開第１０７２９２１６９（ＣＮ，Ａ）
千代 英一郎 外2名，“グラフ縮約に基づくＳＰＡＲＱＬクエリ並列化方法の設計および予備評価”，情報処理学会論文誌 論文誌ジャーナル Ｖｏｌ．５３ Ｎｏ．１２ ［ＣＤ－ＲＯＭ］，日本，一般社団法人情報処理学会，２０１２年１２月１５日，第５３巻，第１２号，ｐ．２８１５－２８２８

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ２１／５５
Ｇ０６Ｆ １１／３４
Ｇ０６Ｆ １６／９０３