(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7517455号
(P7517455)

(45)発行日　令和6年7月17日(2024.7.17)

(24)登録日　令和6年7月8日(2024.7.8)

(51)国際特許分類

Ｇ０６Ｆ　21/12　(2013.01)	Ｇ０６Ｆ	21/12	３５０
Ｇ０６Ｆ　21/64　(2013.01)	Ｇ０６Ｆ	21/64	

ＦＩ

請求項の数　7　（全22頁）

(21)出願番号　特願2022-558652(P2022-558652)
(86)(22)出願日　令和2年10月28日(2020.10.28)
(86)国際出願番号　PCT/JP2020/040338
(87)国際公開番号　WO2022/091231
(87)国際公開日　令和4年5月5日(2022.5.5)
審査請求日　令和5年4月21日(2023.4.21)
（出願人による申告）平成３０年度 国立研究開発法人新エネルギー・産業技術総合開発機構「戦略的イノベーション創造プログラム（ＳＩＰ）第２期／ＩｏＴ社会に対応したサイバー・フィジカル・セキュリティ／（Ａ２）ＩｏＴ機器等向け真贋判定による信頼の証明技術の研究開発事業」に関する研究開発、産業技術力強化法第１７条の適用を受ける特許出願

(73)特許権者　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(74)代理人　100103894
弁理士　家入 健
(72)発明者　早木 悠斗
東京都港区芝五丁目７番１号 日本電気株式会社内
(72)発明者　山垣 則夫
東京都港区芝五丁目７番１号 日本電気株式会社内
審査官　松平 英

最終頁に続く

(54)【発明の名称】　改ざん検知機能組み込み装置、改ざん検知機能組み込み方法、及びプログラム

(57)【特許請求の範囲】

【請求項１】

ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む改ざん検知機能組み込み装置であって、

前記ソフトウェアのソースコードである第１のソースコードを入力する入出力部と、

前記第１のソースコードをビルドして第１のバイナリを生成するビルド部と、

前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成するＣＦＧ生成部と、

前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込み部と、

前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する前記ビルド部と、

前記第２のバイナリに基づいて、第２のＣＦＧを生成する前記ＣＦＧ生成部と、

前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成部と、

前記第２のバイナリ及び前記ホワイトリストを出力する前記入出力部と、を備え、

前記ホワイトリスト作成部は、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び

出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する、
改ざん検知機能組み込み装置。
【請求項２】
前記ホワイトリスト作成部は、
前記第２のＣＦＧ上の前記改ざん検知機能呼び出し関数を順次選択し、
前記選択した改ざん検知機能呼び出し関数を含むノードの全ての子孫ノードのうち、前記改ざん検知機能呼び出し関数を含まないノードを辿り、前記選択した改ざん検知機能呼び出し関数を含むノードから、次の前記改ざん検知機能呼び出し関数を含むノードの直前のノードまでを、前記選択した改ざん検知機能呼び出し関数の前記監視範囲に決定する、
請求項１に記載の改ざん検知機能組み込み装置。
【請求項３】
前記ホワイトリスト作成部は、ファイル形式で前記ホワイトリストを作成し、
前記入出力部は、ファイル形式の前記ホワイトリストを出力する、
請求項１又は２に記載の改ざん検知機能組み込み装置。
【請求項４】
前記ホワイトリスト作成部は、ソースコード形式で前記ホワイトリストを作成し、作成したソースコード形式の前記ホワイトリストを前記第２のソースコードに組み込み、
前記ビルド部は、前記ホワイトリストが組み込まれた前記第２のソースコードをビルドして、前記ホワイトリストが組み込まれた前記第２のバイナリを生成し、
前記入出力部は、前記ホワイトリストが組み込まれた前記第２のバイナリを出力する、
請求項１又は２に記載の改ざん検知機能組み込み装置。
【請求項５】
前記改ざん検知機能組み込み部は、前記ソフトウェアが動作する機器が、ハードウェアによって外部からのアクセスが制限された特定のメモリ領域を有する場合に、前記改ざん検知機能を前記特定のメモリ領域上に配置されるソースコードに組み込む、
請求項１から４のいずれか１項に記載の改ざん検知機能組み込み装置。
【請求項６】
ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む改ざん検知機能組み込み装置が実行する改ざん検知組み込み方法であって、
前記ソフトウェアのソースコードである第１のソースコードを入力する入力ステップと、
前記第１のソースコードをビルドして第１のバイナリを生成する第１のビルドステップと、
前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成する第１のＣＦＧ生成ステップと、
前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込みステップと、
前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する第２のビルドステップと、
前記第２のバイナリに基づいて、第２のＣＦＧを生成する第２のＣＦＧ生成ステップと、
前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成ステップと、
前記第２のバイナリ及び前記ホワイトリストを出力する出力ステップと、を含み、
前記ホワイトリスト作成ステップでは、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する、
改ざん検知機能組み込み方法。

【請求項７】
ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む処理をコンピュータに実行させるプログラムであって、
前記ソフトウェアのソースコードである第１のソースコードを入力する入力ステップと、
前記第１のソースコードをビルドして第１のバイナリを生成する第１のビルドステップと、
前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成する第１のＣＦＧ生成ステップと、
前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込みステップと、
前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する第２のビルドステップと、
前記第２のバイナリに基づいて、第２のＣＦＧを生成する第２のＣＦＧ生成ステップと、
前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成ステップと、
前記第２のバイナリ及び前記ホワイトリストを出力する出力ステップと、を含み、
前記ホワイトリスト作成ステップでは、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する、
プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本開示は、改ざん検知機能組み込み装置、改ざん検知機能組み込み方法、及びコンピュータ可読媒体に関し、特に、ＩｏＴ（Ｉｎｔｅｒｎｅｔ　ｏｆ　Ｔｈｉｎｇｓ）機器などの機器のソフトウェアに用いる改ざん検知機能組み込み装置、改ざん検知機能組み込み方法、及びコンピュータ可読媒体に関する。
【背景技術】
【０００２】
近年、ＩｏＴ機器の普及により、ＩｏＴ機器のようなメモリやＣＰＵ（Ｃｅｎｔｒａｌ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）などのリソースが潤沢ではない機器でも動作するセキュリティシステムが要求されている。
【０００３】
関連するセキュリティシステムとして、ハッシュ値を用いたホワイトリスト型の改ざん検知機能を機器に組み込み、機器上のソフトウェアが正しい状態であるか否か（改ざんされていないか否か）を監視する方式が知られている。
【０００４】
ハッシュ値を用いたホワイトリスト型の改ざん検知機能とは、正常時の機器のメモリの情報を予めホワイトリストに登録しておき、動作中の機器のメモリの情報を、ホワイトリストに登録されているメモリの情報と比較し、改ざんの有無を監視する機能である。このとき、メモリの情報はハッシュ値の形で管理される。
【０００５】
ハッシュ値を用いたホワイトリスト型の改ざん検知機能は、次のように実現される。まず、事前に、正常時の機器のメモリの情報を何等かの方法で取得し、取得したメモリの情報をホワイトリストに登録する。メモリの情報は、実行コードがメモリ上にどのように展開されているのかを示す情報となる。次に、機器の動作中の任意のタイミングでメモリの情報を取得し、取得したメモリの情報をホワイトリストに登録されているメモリの情報と

比較する。比較の結果、機器の動作中に取得したメモリの情報とホワイトリストに登録されているメモリの情報とが一致すれば、改ざんがなく（攻撃を受けていない）、一致しなければ、改ざんされている（攻撃されている）ことになる。

【０００６】

ホワイトリストの意味は、正常時のメモリのスナップショットということになる。ただし、メモリのスナップショットをそのままの形で登録・比較するよりも、ハッシュ値の形で登録・比較した方が、機器本来の動作への影響が小さくなる。そのため、ホワイトリストには、正常時のメモリの情報がハッシュ値の形で登録されており、動作中のメモリの情報もハッシュ値の形で管理される。ハッシュ値を用いたホワイトリスト型の改ざん検知機能の一例が、非特許文献１，２に開示されている。

【０００７】

非特許文献１には、機器に対する入力がトリガーとなり、次に実行する機能の改ざんを検知する方法が開示されている。非特許文献１の方法によれば、ユーザーの入力により、次に実行される機能（ここでは、機能Ａと呼ぶ）が決まる。また、ユーザーの入力により、改ざん検知機能を呼び出す関数（以下、改ざん検知機能呼び出し関数と呼ぶ）が起動する。改ざん検知機能呼び出し関数は、改ざん検知機能を呼び出し、機能Ａの実行時に使用するメモリ領域に限定して、改ざんの有無を監視する。

【０００８】

非特許文献２には、一度の監視における監視範囲を制御フローグラフ（Ｃｏｎｔｒｏｌ Ｆｌｏｗ Ｇｒａｐｈ：ＣＦＧ）の１つのノードとし、非特許文献１の方法よりも高速な監視を目指す方法が開示されている。ここで、ＣＦＧとは、プログラムがどの順序で実行されるかをグラフ化したものである。したがって、ＣＦＧは有向グラフである。ＣＦＧのノードは連続するプログラムの実行コードとなる。実行コードは、バイナリレベルでもソースコードレベルでも記載可能であるが、ここではバイナリレベルで記載されているものを扱う。このとき、ＣＦＧの１つのノードは分岐命令ごとに区切られる。プログラムを分岐命令ごとに区切ったブロックのことベーシックブロックと呼ぶ。すなわち、ＣＦＧの１つのノードは１つのベーシックブロックである、と言い換えることができる。非特許文献２には、このようなＣＦＧを基に、各ノードの実行コードを監視するためのトリガー（上記の改ざん検知機能呼び出し関数に相当）をソースコードに組み込む方法が開示されている。

【０００９】

特許文献１には、バイトコード上に直接改ざん検知機能を組み込む方法が開示されている。特許文献１の方法によれば、改ざん検知機能を組み込む対象のプログラムに対して、関数ごとにＣＦＧを作成する。そして、ＣＦＧを基に関数ごとにランダムにノードを抽出し、抽出したノード及びそれ以前に必ず実行されるノードに改ざん検知機能を組み込む。

【００１０】

特許文献２には、チェッカコード（上記の改ざん検知機能に相当）が改ざん検知処理を行う頻度を制御する方法が開示されている。特許文献２の方法によれば、チェッカコードには活性状態と不活性状態があり、プログラム上に組み込まれたチェッカコードが活性状態のときにのみ、改ざん検知処理を行う。

【先行技術文献】

【特許文献】

【００１１】

【文献】国際公開第２０１８／１５０６１９号
【文献】特開２００８－０８４２７５号公報

【非特許文献】

【００１２】

【文献】Toshiki Kobayashi、Takayuki Sasaki、Astha Jada、Daniele E. Asoni、Adrian Perrig、“SAFES: Sand-boxed Architecture for Frequent Environment Self-measurement”、Proceedings of the 3rd Workshop on System Software for Tr

usted Execution、２０１８年、ｐｐ．３７－４１
【文献】早木　悠斗、佐々木　貴之、リュウ　センペイ、冨田　光輝、山垣　則夫著、「IoT機器向け改ざん検知機能による信頼の証明技術の提案」、ＳＣＩＳ２０２０、２０２０年、ｐｐ．１－６
【発明の概要】
【発明が解決しようとする課題】
【００１３】
　ＩｏＴ機器などの機器に用いる上記の改ざん検知機能においては、機器本来の動作への影響を低減することで、機器の運用への影響を低減する必要がある。機器本来の動作への影響を低減するための方法として、非特許文献１及び非特許文献２の方法のように、一度の監視における監視範囲を絞る、という方法が考えられる。その際、プログラムのうち直後に実行される部分は漏れなく監視される必要がある。また、別の観点として、ＩｏＴ機器などの機器のソフトウェアは、定期的なアップデートが行われる。したがって、実用面を考慮すると、ソフトウェアの開発者が簡単に改ざん検知機能を組み込める必要がある。
【００１４】
　非特許文献１の方法においては、ユーザーの入力をトリガーとするため、ユーザーからの入力が少ない仕様の機器では、改ざん検知機能呼び出し関数による一度の監視範囲が広くなり、結果として機器の動作に影響を及ぼす可能性がある。また、非特許文献１の方法においては、１つの機能の中に命令数の多い関数が存在するような場合においても、改ざん検知機能の稼働が機器本来の動作に影響を与える可能性がある。さらに、非特許文献１には、改ざん検知機能の組み込み方法については明示されていない。
【００１５】
　特許文献１の方法においては、改ざん検知機能をＣＦＧ上のノードの中にランダムに組み込むため、組み込まれるノードの間隔が広くなる可能性がある。また、特許文献１の方法においては、改ざん検知手法に関しては指定していない。これらを考慮すると、特許文献１の方法においては、改ざん検知機能呼び出し関数による一度の監視における監視範囲が広くなり、改ざんの有無を監視する監視時間が機器の動作に影響を与える可能性がある。
【００１６】
　特許文献２の方法においては、チェッカコードが不活性の場合には改ざん検知処理を行わないようにすることで、処理負荷の低減を行っている。この場合、改ざんが起こった直後に該当範囲を監視するチェッカコードが不活性であれば、改ざんを検知せずにプログラムが実行される可能性がある。また、不活性なチェッカコードがあっても、漏れなく改ざんを検知しようとすれば、改ざん検知機能呼び出し関数による一度の監視における監視範囲が広くなると考えられるため、その場合も、一度の監視における監視時間が長くなってしまう可能性がある。
【００１７】
　その一方、非特許文献２の方法においては、改ざん検知機能がプログラムのベーシックブロック単位で実行されるため、非特許文献１の方法とは異なり、ユーザーの入力等に依存せずに、改ざんの有無を監視することができる。また、ベーシックブロックと関数とを比較した場合、一般的にベーシックブロックの方が関数よりも命令数が少なくなる。そのため、非特許文献２の方法においては、改ざん検知機能呼び出し関数による一度の監視における監視範囲が広くなることを回避でき、機器本来の動作への影響を低減できる可能性がある。また、非特許文献２には、改ざん検知機能をソースコードに組み込む方法についても明記されている。
【００１８】
　しかし、非特許文献２の方法においては、特定の条件下では、監視対象とならないベーシックブロックが存在し、監視漏れが発生する可能性がある。監視対象とならないベーシックブロックが存在する条件とは、条件分岐やループ文などの構文がソースコード内に存在することを指す。これらの構文がソースコード内に存在する場合、ソースコード上に改ざん検知機能呼び出し関数を挿入できない箇所が存在する可能性があるため、全てのベー

シックブロックに対応するように改ざん検知機能呼び出し関数を組み込むことができない。このような例を、図１を参照して説明する。

【００１９】

図１は、非特許文献２の方法で監視対象とならないベーシックブロックが存在するソースコード及びＣＦＧの例を示している。図１の右側は、ソースコードを示し、図１の左側は、右側のソースコードをビルドしたバイナリを基に生成されたＣＦＧを示している。また、図１の左側の各ノード内の数字は、右側のソースコードの行番号を示しており、各ノードには、各ノード内の数字で示される行番号に相当する実行コードが含まれることを表している。例えば、ノード２には、ソースコードの６～７行目に相当する実行コードが含まれる。図１の例では、ノード２を監視するトリガーは、５行目と６行目の間に挿入できると考えられる。しかし、ノード３は、８行目に相当する実行コードが含まれており、８行目を監視するためにトリガーをソースコード上に挿入することができず、監視対象外となってしまう。

【００２０】

以上の通り、非特許文献１及び特許文献１，２の方法においては、改ざん検知機能呼び出し関数による監視範囲が広くなってしまう可能性があるという課題がある。その一方、非特許文献２の方法においては、改ざん検知機能呼び出し関数による監視範囲が広くなることを回避できる可能性があるものの、監視対象とならないノード（ベーシックブロック）が存在し、監視漏れが発生する可能性があるという課題がある。

【００２１】

そこで本開示の目的は、上述した課題を鑑み、上述した課題のいずれかを解決することができる改ざん検知機能組み込み装置、改ざん検知機能組み込み方法、及びコンピュータ可読媒体を提供することにある。

【課題を解決するための手段】

【００２２】

本開示の一態様に係る改ざん検知機能組み込み装置は、

ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む改ざん検知機能組み込み装置であって、

前記ソフトウェアのソースコードである第１のソースコードを入力する入出力部と、

前記第１のソースコードをビルドして第１のバイナリを生成するビルド部と、

前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成するＣＦＧ生成部と、

前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込み部と、

前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する前記ビルド部と、

前記第２のバイナリに基づいて、第２のＣＦＧを生成する前記ＣＦＧ生成部と、

前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成部と、

前記第２のバイナリ及び前記ホワイトリストを出力する前記入出力部と、を備え、

前記ホワイトリスト作成部は、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する。

【００２３】

本開示の他の態様に係る改ざん検知機能組み込み方法は、

ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む改ざん検知機能組み込み装置が実行する改ざん検知組み込み方法であって、

前記ソフトウェアのソースコードである第１のソースコードを入力する入力ステップと、
前記第１のソースコードをビルドして第１のバイナリを生成する第１のビルドステップと、
前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成する第１のＣＦＧ生成ステップと、
前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込みステップと、
前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する第２のビルドステップと、
前記第２のバイナリに基づいて、第２のＣＦＧを生成する第２のＣＦＧ生成ステップと、
前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成ステップと、
前記第２のバイナリ及び前記ホワイトリストを出力する出力ステップと、を含み、
前記ホワイトリスト作成ステップでは、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する。

【００２４】

本開示のさらに他の態様に係るコンピュータ可読媒体は、
ハッシュ値を用いたホワイトリスト型の改ざん検知機能を、監視対象のソフトウェアに組み込む処理をコンピュータに実行させるプログラムが格納された非一時的なコンピュータ可読媒体であって、
前記プログラムは、
前記ソフトウェアのソースコードである第１のソースコードを入力する入力ステップと、
前記第１のソースコードをビルドして第１のバイナリを生成する第１のビルドステップと、
前記第１のバイナリに基づいて、第１の制御フローグラフ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｇｒａｐｈ；ＣＦＧ）を生成する第１のＣＦＧ生成ステップと、
前記第１のＣＦＧに基づいて、改ざん検知機能呼び出し関数を前記第１のソースコードに組み込む組み込み箇所を決定し、決定した前記第１のソースコードの組み込み箇所に前記改ざん検知機能呼び出し関数を組み込むと共に、前記第１のソースコードに前記改ざん検知機能を組み込む改ざん検知機能組み込みステップと、
前記第１のソースコードに前記改ざん検知機能及び前記改ざん検知機能呼び出し関数が組み込まれたソースコードである第２のソースコードをビルドして第２のバイナリを生成する第２のビルドステップと、
前記第２のバイナリに基づいて、第２のＣＦＧを生成する第２のＣＦＧ生成ステップと、
前記第２のバイナリ及び前記第２のＣＦＧに基づいて、ホワイトリストを作成するホワイトリスト作成ステップと、
前記第２のバイナリ及び前記ホワイトリストを出力する出力ステップと、を含み、
前記ホワイトリスト作成ステップでは、前記第２のＣＦＧに基づいて、前記改ざん検知機能呼び出し関数の監視範囲を決定し、前記改ざん検知機能呼び出し関数の前記監視範囲のハッシュ値のリストを前記ホワイトリストとして作成する。

【発明の効果】

【００２５】

上述の態様によれば、監視対象のソフトウェアにおいて、監視できないＣＦＧ上のノードの数を低減できるという効果が得られる。

【図面の簡単な説明】

【００２６】

【図１】ソースコード及びＣＦＧの例を示す図である。
【図２】実施の形態１に係る改ざん検知機能組み込み装置の構成例を示すブロック図である。
【図３】実施の形態１に係る改ざん検知機能組み込み装置の全体の動作例を説明する流れ図である。
【図４Ａ】実施の形態１に係るホワイトリスト作成部が図３のステップＡ７で行う動作例を説明する流れ図である。
【図４Ｂ】実施の形態１に係るホワイトリスト作成部が図３のステップＡ７で行う動作例を説明する流れ図である。
【図５】図１のソースコードに対して、図３のステップＡ６までの処理を行い、改ざん検知機能呼び出し関数を組み込んだソースコード及びＣＦＧの例を示す図である。
【図６】実施の形態２に係る改ざん検知機能組み込み装置の構成例を示すブロック図である。
【図７】実施の形態２に係る改ざん検知機能組み込み装置の全体の動作例を説明する流れ図である。
【図８】実施の形態３に係る改ざん検知機能組み込み装置の構成例を示すブロック図である。
【図９】実施の形態４に係る改ざん検知機能組み込み装置のハードウェア構成例を示すブロック図である。
【発明を実施するための形態】
【００２７】
以下、図面を参照しつつ、本開示の実施の形態について説明する。なお、以下の実施の形態において、同一又は同等の要素には、同一の符号を付し、重複する説明は省略される。また、以下の各実施の形態で説明する改ざん検知機能組み込み装置は、監視対象のソフトウェアに対して、ハッシュ値を用いたホワイトリスト型の改ざん検知機能を組み込む装置の例である。
【００２８】
［実施の形態１］
［実施の形態１の構成］
まず、図２を参照して、本実施の形態１に係る改ざん検知機能組み込み装置１００の構成例について説明する。なお、図２において、一方向性の矢印は、あるデータ（又は、信号、情報等）の流れの方向を端的に示したもので、双方向性を排除するものではない（後述する図６及び図８において同じ）。
【００２９】
図２を参照すると、本実施の形態１に係る改ざん検知機能組み込み装置１００は、入出力部１０１と、ビルド部１０２と、Ｃｏｎｔｒｏｌ Ｆｌｏｗ Ｇｒａｐｈ（ＣＦＧ）生成部１０３と、改ざん検知機能組み込み部１０４と、ホワイトリスト作成部１０５と、記憶部１０６と、を含む。
【００３０】
これらの構成要素は、それぞれ次のように動作する。
入出力部１０１には、監視対象のソフトウェア、すなわち、改ざん検知機能を組み込む対象となるソフトウェアのソースコード（第１のソースコード）が入力される。監視対象のソフトウェアは、例えば、ＩｏＴ機器上の制御ソフトウェア等である。入出力部１０１は、入力されたソースコードを記憶部１０６に記憶させ、ビルド部１０２に処理を行うよう通知を送る。
【００３１】
また、入出力部１０１は、ホワイトリスト作成部１０５から後述する通知を受け取ると、記憶部１０６から、後述する改ざん検知機能及び改ざん検知機能呼び出し関数を組み込んだソフトウェアのバイナリ（第２のバイナリ。以下、改ざん検知機能組み込みバイナリと呼ぶ）を読み出し、読み出した改ざん検知機能組み込みバイナリを出力する。また、入

出力部１０１は、記憶部１０６から、後述するファイル形式のホワイトリストを読み出し、読み出したホワイトリストを改ざん検知機能組み込みバイナリと共に出力する。ただし、入出力部１０１は、記憶部１０６から改ざん検知機能組み込みバイナリ及びホワイトリストを読み出すことには限定されない。入出力部１０１は、改ざん検知機能組み込みバイナリをビルド部１０２から受け取ってもよいし、ホワイトリストをホワイトリスト作成部１０５から受け取ってもよい。

【００３２】

ビルド部１０２は、入出力部１０１から通知を受け取ると、記憶部１０６からソースコードを読み出し、ビルドを行う。ビルド部１０２は、ビルドにより生成されたバイナリ（第１のバイナリ）を記憶部１０６に記憶させ、ＣＦＧ生成部１０３に処理を行うよう通知を送る。

【００３３】

また、ビルド部１０２は、改ざん検知機能組み込み部１０４から後述する通知を受け取ると、記憶部１０６から、後述する改ざん検知機能及び改ざん検知機能呼び出し関数を組み込んだソースコード（第２のソースコード。以下、改ざん検知機能組み込みソースコードと呼ぶ）を読み出し、ビルドを行う。ビルド部１０２は、ビルドにより生成された改ざん検知機能組み込みバイナリを記憶部１０６に記憶させ、ＣＦＧ生成部１０３に処理を行うよう通知を送る。

【００３４】

なお、ビルド部１０２は、ＣＦＧ生成部１０３に通知を送る際、ＣＦＧ生成部１０３が処理を行うバイナリを指定する。また、ビルド部１０２は、入出力部１０１及び改ざん検知機能組み込み部１０４のどちらから通知を受けたかによって、ビルドにより生成したバイナリに改ざん検知機能が組み込まれているか否かを判断できる。そのため、ビルド部１０２は、ＣＦＧ生成部１０３が処理を行うバイナリに改ざん検知機能が組み込まれているか否かの情報もＣＦＧ生成部１０３に通知する。

【００３５】

ＣＦＧ生成部１０３は、ビルド部１０２から通知を受け取ると、その通知で指定されたバイナリを記憶部１０６から読み出し、読み出したバイナリを解析し、ＣＦＧを生成する。ＣＦＧ生成部１０３は、当該バイナリに改ざん検知機能が組み込まれていない場合は、当該バイナリから生成したＣＦＧ（第１のＣＦＧ）を改ざん検知機能組み込み部１０４に送り、当該バイナリが、改ざん検知機能が組み込まれた改ざん検知機能組み込みバイナリである場合は、当該バイナリから生成したＣＦＧ（第２のＣＦＧ）をホワイトリスト作成部１０５に送る。

【００３６】

改ざん検知機能組み込み部１０４は、ＣＦＧ生成部１０３からＣＦＧを受け取ると、当該ＣＦＧを基に、改ざん検知機能呼び出し関数を組み込むソースコードの組み込み箇所を決定し、決定した組み込み箇所に改ざん検知機能呼び出し関数を組み込む。改ざん検知機能組み込み部１０４は、改ざん検知機能もソースコードに組み込む。このようにして改ざん検知機能呼び出し関数及び改ざん検知機能が組み込まれたソースコード一式が改ざん検知機能組み込みソースコードである。改ざん検知機能組み込み部１０４は、改ざん検知機能組み込みソースコードを記憶部１０６に記憶させ、ビルド部１０２に処理を行うよう通知を送る。

【００３７】

ホワイトリスト作成部１０５は、ＣＦＧ生成部１０３からＣＦＧを受け取ると、当該ＣＦＧを基に、記憶部１０６から改ざん検知機能組み込みバイナリを読み出す。ホワイトリスト作成部１０５は、当該ＣＦＧ及び当該改ざん検知機能組み込みバイナリを基に、ホワイトリストをファイル形式で作成し、作成したホワイトリストを記憶部１０６に記憶させる。また、ホワイトリスト作成部１０５は、入出力部１０１に処理を行うよう通知を送る。

【００３８】

記憶部１０６は、入出力部１０１から受け取ったソースコード、改ざん検知機能組み込

み部１０４から受け取った改ざん検知機能組み込みソースコード、ビルド部１０２から受け取ったバイナリ及び改ざん検知機能組み込みバイナリ、及びホワイトリスト作成部１０５から受け取ったホワイトリストを記憶する。

【００３９】

なお、記憶部１０６は、改ざん検知機能組み込み装置１００において必須の構成要素ではなく、改ざん検知機能組み込み装置１００の外部に設けてもよい。すなわち、改ざん検知機能組み込み装置１００は、入出力部１０１と、ビルド部１０２と、ＣＦＧ生成部１０３と、改ざん検知機能組み込み部１０４と、ホワイトリスト作成部１０５と、からなる最小構成で実現してもよい。

【００４０】

［実施の形態１の動作］

次に、図３の流れ図を参照して、本実施の形態１に係る改ざん検知機能組み込み装置１００の全体の動作例について説明する。

【００４１】

図３を参照すると、まず、入出力部１０１には、監視対象のソフトウェアのソースコードであって、ビルドに必要な全てのソースコードが入力される。入出力部１０１は、入力されたソースコードを記憶部１０６に記憶させ、ビルド部１０２に処理を行うよう通知を送る（ステップＡ１）。

【００４２】

ビルド部１０２は、入出力部１０１から通知を受け取ると、記憶部１０６から、ステップＡ１で入力されたソースコードを読み出し、読み出したソースコードをビルドし、バイナリを生成する。ビルド部１０２は、生成したバイナリを記憶部１０６に記憶させ、ＣＦＧ生成部１０３に処理を行うよう通知を送る（ステップＡ２）。

【００４３】

ＣＦＧ生成部１０３は、ビルド部１０２から通知を受け取ると、記憶部１０６から、ステップＡ２で生成されたバイナリを読み出し、読み出したバイナリを解析し、ＣＦＧを生成する。さらに、ＣＦＧ生成部１０３は、生成したＣＦＧを改ざん検知機能組み込み部１０４へ送る（ステップＡ３）。

【００４４】

改ざん検知機能組み込み部１０４は、ＣＦＧ生成部１０３から、ステップＡ３で生成されたＣＦＧを受け取ると、受け取ったＣＦＧを基に、改ざん検知機能呼び出し関数を組み込むソースコードの組み込み箇所を決定し、決定した組み込み箇所に改ざん検知機能呼び出し関数を組み込む。さらに、改ざん検知機能組み込み部１０４は、改ざん検知機能もソースコードに組み込む。これにより、改ざん検知機能組み込み部１０４は、改ざん検知機能呼び出し関数及び改ざん検知機能を組み込んだ改ざん検知機能組み込みソースコードを生成する。さらに、改ざん検知機能組み込み部１０４は、生成した改ざん検知機能組み込みソースコードを記憶部１０６に記憶させ、ビルド部１０２に処理を行うよう通知を送る（ステップＡ４）。

【００４５】

ビルド部１０２は、改ざん検知機能組み込み部１０４から通知を受け取ると、記憶部１０６から、ステップＡ４で生成された改ざん検知機能組み込みソースコードを読み出し、読み出した改ざん検知機能組み込みソースコードをビルドし、改ざん検知機能組み込みバイナリを生成する。ビルド部１０２は、生成した改ざん検知機能組み込みバイナリを記憶部１０６に記憶させ、ＣＦＧ生成部１０３に処理を行うよう通知を送る（ステップＡ５）。

【００４６】

ＣＦＧ生成部１０３は、ビルド部１０２から通知を受け取ると、記憶部１０６から、ステップＡ５で生成された改ざん検知機能組み込みバイナリを読み出し、読み出した改ざん検知機能組み込みバイナリを解析し、ＣＦＧを生成する。さらに、ＣＦＧ生成部１０３は、生成したＣＦＧをホワイトリスト作成部１０５に送る（ステップＡ６）。

【００４７】

ホワイトリスト作成部１０５は、ＣＦＧ生成部１０３から、ステップＡ６で生成されたＣＦＧを受け取ると、受け取ったＣＦＧを基に、記憶部１０６から改ざん検知機能組み込みバイナリを読み出す。さらに、ホワイトリスト作成部１０５は、ＣＦＧ及び改ざん検知機能組み込みバイナリを基に、ホワイトリストをファイル形式で作成する。さらに、ホワイトリスト作成部１０５は、ホワイトリストを記憶部１０６に記憶させ、入出力部１０１に処理を行うよう通知を送る（ステップＡ７）。なお、ステップＡ７の詳細については後述する。

【００４８】

入出力部１０１は、ホワイトリスト作成部１０５から通知を受け取ると、記憶部１０６から、ステップＡ５で生成された改ざん検知機能組み込みバイナリ及びステップＡ７で作成されたホワイトリストを読み出し、読み出した改ざん検知機能組み込みバイナリ及びホワイトリストを出力する（ステップＡ８）。

【００４９】

次に、図４Ａ及び図４Ｂの流れ図を参照して、図３のステップＡ７でホワイトリスト作成部１０５が行う動作について詳細に説明する。ここでは、図５も参照して説明を行う。図５は、図１のソースコードに対して、図３のステップＡ６までの処理を行い、改ざん検知機能呼び出し関数を組み込んだソースコード及びＣＦＧの例を示している。また、図５の右側及び左側の関係は、図１の右側及び左側の関係と同じである。

【００５０】

図４Ａ及び図４Ｂを参照すると、ホワイトリスト作成部１０５は、ＣＦＧ生成部１０３から、ステップＡ６で生成されたＣＦＧを受け取ると、まず、受け取ったＣＦＧがループを持つか否かを判断する（ステップＢ１）。ここで、ＣＦＧがループを持つ場合とは、プログラム中にループ文（例えば、Ｃ言語におけるｆｏｒ文、ｗｈｉｌｅ文など）が存在することを意味している。逆に、ＣＦＧがループを持たない場合とは、プログラム中にループ文が存在しないことを意味していると言える。後述するステップＢ６以降の処理では、ＣＦＧがループを持たない形にする必要がある。そのため、ＣＦＧがループを持つ場合（ステップＢ１がＹＥＳの場合）は、ステップＢ２へ進み、ステップＢ２でＣＦＧからループを取り除く処理が行われる。一方、ＣＦＧがループを持たない場合（ステップＢ１がＮＯの場合）は、ステップＢ２をスキップして、ステップＢ３へ進む。

【００５１】

ホワイトリスト作成部１０５は、ステップＢ１でＣＦＧがループを持っていた場合、全てのループに対して、アドレス値の数字が大きい方から小さい方に伸びているエッジを削除する操作を行う（ステップＢ２）。ＣＦＧには、実行コードとその実行コードがメモリ上のどのアドレスに存在するかを示す情報が含まれている。そのため、ホワイトリスト作成部１０５は、その情報を用いることで、上記の操作を実現できる。

【００５２】

次に、ホワイトリスト作成部１０５は、空のホワイトリスト（図４Ａ及び図４ＢのステップＢ４以降では、リストと呼ぶ）を作成する（ステップＢ３）。ホワイトリストには、以下のステップＢ４～Ｂ１３の作業を行うことで要素を追加する。

【００５３】

まず、ホワイトリスト作成部１０５は、ステップＡ６で生成されたＣＦＧの中に、ホワイトリストに追加されていない改ざん検知機能呼び出し関数（ここでは、フックと呼ぶ）が存在するか否かを判断する（ステップＢ４）。該当するフックが存在する場合（ステップＢ４がＹＥＳの場合）は、ステップＢ５に進み、該当するフックが存在しない場合（ステップＢ４がＮＯの場合）は、ホワイトリスト作成部１０５は、入出力部１０１へ通知を送り、ステップＡ７の処理を終了する。

【００５４】

ホワイトリスト作成部１０５は、ステップＢ４で該当するフックが存在する場合（ステップＢ４がＹＥＳの場合）、該当するフックの中から１つのフックを選択する（ステップＢ５）。

以後のステップＢ６～Ｂ１１の処理において、ステップＢ５で選択した当該フックが監視する監視範囲を確定する。ここでは、ホワイトリスト作成部１０５が、当該フックとして図５のフックＨ１を選択した場合を例に挙げて説明を行う。

【００５５】

次に、ホワイトリスト作成部１０５は、当該フックを含むノードの当該フックより後の実行コードに相当するアドレス範囲を、当該フックの監視範囲に追加する（ステップＢ６）。当該フックとしてフックＨ１を選択した例では、当該フックを含むノードとは、ノード１のことであり、当該フックを含むノードの当該フックより後の実行コードに相当するアドレス範囲とは、ソースコードの２行目から５行目のことである。なお、本例ではフックＨ１はノード１の先頭に位置しているが、フックはノードの任意の場所にあってもよい。その場合も当該フックを含むノードの当該フックより後の実行コードに相当するアドレス範囲を、当該フックの監視範囲に追加する。

【００５６】

次に、ホワイトリスト作成部１０５は、ステップＢ５で選択した当該フックを含むノードを先頭とする部分グラフを取り出す（ステップＢ７）。この操作は、木から当該ノードを根とする部分木を選ぶ操作と同等である（ただし、ＣＦＧには閉路が存在するため、厳密には木ではない）。当該フックとしてフックＨ１を選択した例では、ホワイトリスト作成部１０５は、ノード１を先頭とする部分グラフを取り出す。この例では、当該部分グラフには、ノード１以下の全てのノードが含まれる。

【００５７】

次に、ホワイトリスト作成部１０５は、ステップＢ７で取り出した当該部分グラフの中に、当該フックを含むノード以外のノードが存在するか否か、すなわち、当該フックを含むノードが子ノードを持つか否かを判断する（ステップＢ８）。当該フックを含むノードが子ノードを持つ場合（ステップＢ８がＹＥＳの場合）は、ステップＢ９へ進み、子ノードを持たない場合（ステップＢ８がＮＯの場合）は、ステップＢ１２へ進む。当該フックとしてフックＨ１を選択した例では、フックＨ１を含むノード１は、子ノード（ノード３、及び、フックＨ２を含むノード２）を持っている。そのため、この例では、ステップＢ９へ進む。

【００５８】

ステップＢ８で、当該フックを含むノードが子ノードを持つ場合（ステップＢ８がＹＥＳの場合）、ホワイトリスト作成部１０５は、当該部分グラフの中に、当該フック以外のフックが存在するか否かを判断する（ステップＢ９）。当該部分グラフの中に、当該フック以外のフックが存在する場合（ステップＢ９がＹＥＳの場合）は、ステップＢ１１へ進み、当該フック以外のフックが存在しない場合（ステップＢ９がＮＯの場合）は、ステップＢ１０へ進む。当該フックとしてフックＨ１を選択した例では、当該部分グラフの中に、フックＨ１以外に、フックＨ２，Ｈ４，Ｈ５，Ｈ６が存在している。そのため、この例では、ステップＢ１１へ進む。

【００５９】

ステップＢ９で、当該部分グラフの中に、当該フック以外のフックが存在しない場合（ステップＢ９がＮＯの場合）、ホワイトリスト作成部１０５は、当該部分グラフから、当該フックを含むノードを取り除いた部分に相当するアドレス範囲を、当該フックの監視範囲に追加する（ステップＢ１０）。当該フックを含むノードを、当該フックの監視範囲から取り除くのは、ステップＢ６で当該フックを含むノードをすでに監視範囲に追加しているからである。その後、ステップＢ１２へ進む。ホワイトリスト作成部１０５は、当該フックとしてフックＨ１を選択したときに、仮に、図５のフックＨ２，Ｈ４，Ｈ５，Ｈ６が存在しなかったとすれば、ノード２，３，４，５，６の実行コードに相当するアドレス範囲を、フックＨ１の監視範囲に追加する。

【００６０】

一方、ステップＢ９で、当該部分グラフの中に、当該フック以外のフックが存在する場合（ステップＢ９がＹＥＳの場合）、ホワイトリスト作成部１０５は、当該部分グラフの

うち、当該フックを含むノードを先頭ノードとし、フックのないノードのみで構成された部分グラフを取り出す。そして、ホワイトリスト作成部１０５は、取り出した部分グラフから、当該フックを含むノードを取り除いた部分に相当するアドレス範囲（範囲Ａ）を、当該フックの監視範囲に追加する。また、当該フック以外のフックがノードの先頭にない場合、当該フック以外のフックが存在するノードであり、かつ、当該フックがあるノードあるいは範囲Ａと直接つながっているノードに関して、フックより前の範囲を範囲Ａに追加する（ステップＢ１１）。当該フックを含むノードを範囲Ａから取り除くのは、ステップＢ６で当該フックを含むノードをすでに監視範囲に追加しているからである。その後、ステップＢ１２へ進む。当該フックとしてフックＨ１を選択した例では、フックＨ１を含むノード１を先頭ノードとする部分グラフの中で、フックを含まないノードはノード３のみである。そのため、この例では、範囲Ａに相当するのはノード３となる。したがって、ノード３の実行コードに相当するアドレス範囲を、フックＨ１の監視範囲に追加する。また、仮に、ノード５がフックを含むノードではなかった場合、範囲Ａはノード３及びノード５の実行コードに相当するアドレス範囲となる。別の例として、ノード５のフックＨ５がノード５の途中にあった場合、ノード５の先頭からフックＨ５の前までの範囲の実行コードのアドレス範囲を範囲Ａに追加する。

【００６１】

以上のステップＢ６～Ｂ１１の処理により、ステップＢ５で選択した当該フックの監視範囲が確定する。当該フックとしてフックＨ１を選択した例では、フックＨ１の監視範囲は、ノード１及びノード３の実行コードに相当するアドレス範囲となる。

【００６２】

次に、ホワイトリスト作成部１０５は、ステップＢ６～Ｂ１１の処理により確定した当該フックの監視範囲のハッシュ値を計算する（ステップＢ１２）。当該フックとしてフックＨ１を選択した例では、フックＨ１の監視範囲がノード１及びノード３の実行コードに相当するアドレス範囲となったため、ホワイトリスト作成部１０５は、このアドレス範囲のハッシュ値を計算する。

【００６３】

その後、ホワイトリスト作成部１０５は、当該フックのフックＩＤ、当該フックの監視範囲、及び当該監視範囲のハッシュ値の組をホワイトリストに登録する（ステップＢ１３）。ステップＢ１３の処理が終了すると、ステップＢ４に戻り、ホワイトリスト作成部１０５は、ステップＢ５～Ｂ１３の処理を、ステップＡ６で生成されたＣＦＧの中の全てのフックに対して実行するまで繰り返す。以上の説明では、一例として、フックＨ１を選択したが、ホワイトリスト作成部１０５は、ＣＦＧの中のフックＨ２，Ｈ４，Ｈ５，Ｈ６に対しても同様に処理を行い、監視範囲を決定する。

【００６４】

以上の図４Ａ及び図４Ｂのホワイトリスト作成部１０５の動作のうち、ＣＦＧ上の各フックの監視範囲を決定するまでの動作を要約すると、以下のようになる。ホワイトリスト作成部１０５は、ＣＦＧ上のフックを順次選択し、選択したフックの監視範囲を決定していく。このとき、ホワイトリスト作成部１０５は、選択したフックを含むノードの全ての子孫ノードのうち、フックを含まないノードを辿っていく。そして、ホワイトリスト作成部１０５は、選択したフックを含むノードから、次のフックを含むノードの直前のノードまでを、選択したフックの監視範囲に決定する。

【００６５】

［実施の形態１の効果］

次に、本実施の形態１に係る改ざん検知機能組み込み装置１００の効果について説明する。

本実施の形態１によれば、ホワイトリスト作成部１０５は、ホワイトリストの作成において、ＣＦＧを基に、改ざん検知機能呼び出し関数の監視範囲を決定する。そのため、改ざん検知機能呼び出し関数の監視範囲を適切に決定できる。その結果、監視対象のソフトウェアにおいて、監視できないＣＦＧ上のノードの数を低減できる。

【００６６】
より具体的には、本実施の形態１によれば、ホワイトリスト作成部１０５は、ＣＦＧ上の改ざん検知機能呼び出し関数を順次選択し、選択した改ざん検知機能呼び出し関数を含むノードの全ての子孫ノードのうち、改ざん検知機能呼び出し関数を含まないノードを辿っていく。そして、ホワイトリスト作成部１０５は、選択した改ざん検知機能呼び出し関数を含むノードから、次の改ざん検知機能呼び出し関数を含むノードの直前のノードまでを、選択した改ざん検知機能呼び出し関数の監視範囲に決定する。そのため、選択した改ざん検知機能呼び出し関数の監視範囲には、選択した改ざん検知機能呼び出し関数を含むノード１つだけではなく、改ざん検知機能呼び出し関数を含まない子孫ノードも含めることができる。その結果、監視対象のソフトウェアにおいて、監視できないＣＦＧ上のノードの数を低減できる。
【００６７】
また、本実施の形態１によれば、改ざん検知機能呼び出し関数は、ＣＦＧ上のノード単位で監視を行うため、一度の監視範囲が広くなることを回避することができる。その結果、一度の監視における監視時間が低減できるため、監視時間が、改ざん検知機能を組み込む機器の動作に影響を与える可能性を低減することが期待できる。
【００６８】
さらに、本実施の形態１によれば、ホワイトリスト作成部１０５は、ＣＦＧ上のノードを辿っていくだけで、改ざん検知機能呼び出し関数の監視範囲を決定できるため、機械的な処理を行うことができる。
【００６９】
［実施の形態２］
上述した実施の形態１に係る改ざん検知機能組み込み装置１００は、ホワイトリストをファイル形式で出力する構成であった。
これに対して、本実施の形態２に係る改ざん検知機能組み込み装置２００は、ホワイトリストをソフトウェア内に組み込んで出力する構成である。
【００７０】
［実施の形態２の構成］
まず、図６を参照して、本実施の形態２に係る改ざん検知機能組み込み装置２００の構成例について説明する。
図６を参照すると、本実施の形態２に係る改ざん検知機能組み込み装置２００は、上述した実施の形態１に係る改ざん検知機能組み込み装置１００における入出力部１０１、ビルド部１０２、及びホワイトリスト作成部１０５を、それぞれ、入出力部２０１、ビルド部２０２、及びホワイトリスト作成部２０５に置き換えたものである。改ざん検知機能組み込み装置２００は、それ以外の構成については、改ざん検知機能組み込み装置１００と同じであるため、詳細な説明は省略する。
【００７１】
上述した実施の形態１に係る入出力部１０１は、ホワイトリスト作成部１０５から通知を受け取ると、記憶部１０６から改ざん検知機能組み込みバイナリ及びホワイトリストを読み出し、出力していた。
これに対して、本実施の形態２に係る入出力部２０１は、ビルド部２０２から後述する通知を受け取ると、記憶部１０６から、後述する、ホワイトリストをさらに組み込んだ改ざん検知機能組み込みバイナリ（以下、ホワイトリスト／改ざん検知機能組み込みバイナリと呼ぶ）を読み出し、出力する。入出力部２０１は、それ以外の動作については、入出力部１０１と同様であるため、詳細な説明は省略する。
【００７２】
本実施の形態２に係るビルド部２０２は、上述した実施の形態１に係るビルド部１０２の動作に加え、次の動作を行う。ビルド部２０２は、ホワイトリスト作成部２０５から後述する通知を受け取ると、記憶部１０６から改ざん検知機能組み込みソースコードを読み出し、ビルドを行う。このときの改ざん検知機能組み込みソースコードには、後述するよ

うに、ソースコード形式のホワイトリストが組み込まれている。そのため、ここでビルド部２０２のビルドにより生成されたバイナリが、ホワイトリスト／改ざん検知機能組み込みバイナリとなる。ビルド部２０２は、生成したホワイトリスト／改ざん検知機能組み込みバイナリを記憶部１０６に記憶させ、入出力部２０１に処理を行うよう通知を送る。

【００７３】

上述した実施の形態１に係るホワイトリスト作成部１０５は、ホワイトリストをファイル形式で作成し、作成したホワイトリストを記憶部１０６に記憶させ、入出力部１０１に処理を行うよう通知を送っていた。

これに対して、本実施の形態２に係るホワイトリスト作成部２０５は、ホワイトリストをソースコード形式で作成する。ホワイトリスト作成部２０５は、作成したホワイトリストを、改ざん検知組み込みソースコードの一部として、記憶部１０６に記憶させ、ビルド部２０２に処理を行うよう通知を送る。

【００７４】

［実施の形態２の動作］

次に、図７の流れ図を参照して、本実施の形態２に係る改ざん検知機能組み込み装置２００の全体の動作例について詳細に説明する。

本実施の形態２に係る改ざん検知機能組み込み装置２００の動作は、以下の動作を除き、上述した実施の形態１に係る改ざん検知機能組み込み装置２００の図３の動作と同じである。

【００７５】

ステップＡ７においては、ホワイトリスト作成部２０５は、ホワイトリストをソースコード形式で作成する。ホワイトリスト作成部２０５は、ソースコード形式のホワイトリストを、改ざん検知組み込みソースコードの一部として、記憶部１０６に記憶させ、ビルド部２０２に処理を行うよう通知を送る。

【００７６】

ステップＡ７の次に行われるステップＡ９においては、ビルド部２０２は、ホワイトリスト作成部２０５から通知を受け取ると、記憶部１０６から改ざん検知機能組み込みソースコードを読み出し、ビルドを行う。このときの改ざん検知機能組み込みソースコードには、ステップＡ７でソースコード形式のホワイトリストが組み込まれている。そのため、ここでビルド部２０２のビルドにより生成されたバイナリが、ホワイトリスト／改ざん検知機能組み込みバイナリとなる。ビルド部２０２は、生成したホワイトリスト／改ざん検知機能組み込みバイナリを記憶部１０６に記憶させ、入出力部２０１に処理を行うよう通知を送る。

【００７７】

ステップＡ９の次に行われるステップＡ８においては、入出力部２０１は、ビルド部２０２から通知を受け取ると、記憶部１０６から、ホワイトリスト／改ざん検知機能組み込みバイナリを読み出し、読み出したホワイトリスト／改ざん検知機能組み込みバイナリを出力する。

【００７８】

［実施の形態２の効果］

次に、本実施の形態２に係る改ざん検知機能組み込み装置２００の効果について説明する。

本実施の形態２によれば、ビルド部２０２は、ソースコード形式のホワイトリストが組み込まれた改ざん検知機能組み込みソースコードのビルドを行う。入出力部２０１は、ビルドにより生成されたホワイトリスト／改ざん検知機能組み込みバイナリを出力する。このように、ホワイトリストをソフトウェアに組み込んでいるため、ファイル入出力が行えないような機器に組み込むソフトウェアに対しても、本開示は適用可能となる。

本実施の形態２は、それ以外の効果は、上述した実施の形態１と同じである。

【００７９】

［実施の形態３］

本実施の形態３に係る改ざん検知機能組み込み装置３００は、Ｔｒｕｓｔｅｄ　Ｅｘｅｃｕｔｉｏｎ　Ｅｎｖｉｒｉｏｎｍｅｎｔ（ＴＥＥ）を用いた構成である。ＴＥＥとは、外部からのメモリの一部領域へのアクセスを、ハードウェアによって制限する機能である。ここでは、外部からのアクセスが制限されたメモリ領域をＳｅｃｕｒｅ　Ｗｏｒｌｄ（ＳＷ）と呼び、それ以外のメモリ領域をＮｏｒｍａｌ　Ｗｏｒｌｄ（ＮＷ）と呼ぶ。

本実施の形態３に係る改ざん検知機能組み込み装置３００では、改ざん検知機能をＳＷ側にソースコードとして組み込むことで、より安全に実行中の機器の改ざんを監視することができる。

【００８０】

［実施の形態３の構成］

まず、図８を参照して、本実施の形態３に係る改ざん検知機能組み込み装置３００の構成例について説明する。

図８を参照すると、本実施の形態３に係る改ざん検知機能組み込み装置３００は、上述した実施の形態２に係る改ざん検知機能組み込み装置２００における改ざん検知機能組み込み部１０４を、改ざん検知機能組み込み部３０４に置き換えたものである。改ざん検知機能組み込み装置３００は、それ以外の構成については、改ざん検知機能組み込み装置２００と同じであるため、詳細な説明は省略する。

【００８１】

本実施の形態３に係る改ざん検知機能組み込み部３０４は、改ざん検知機能をＳＷ側にソースコードとして組み込む。改ざん検知機能組み込み部３０４は、それ以外の動作については、改ざん検知機能組み込み部１０４と同様であるため、詳細な説明は省略する。

【００８２】

［実施の形態３の動作］

次に、本実施の形態３に係る改ざん検知機能組み込み装置３００の全体の動作例について詳細に説明する。

本実施の形態３に係る改ざん検知機能組み込み装置３００の動作は、以下の動作を除き、上述した実施の形態２に係る改ざん検知機能組み込み装置２００の図７の動作と同じである。

【００８３】

ビルド部２０２は、上述したように、図７のステップＡ２、ステップＡ５、及びステップＡ９において、ビルドを行う。その際に、ビルド部２０２は、まず、ＳＷ側に関するビルドを行い、次に、ＮＷ側に関するビルドを行う。ステップＡ２及びステップＡ５においては、ビルド部２０２は、監視対象のソフトウェアの設計に応じて、以下の２つのパターンのいずれかでビルドを行うことが考えられる。

【００８４】

第１のパターンは、監視対象のソフトウェアがＮＷのみで構成されている場合である。この場合は、改ざん検知機能組み込み部３０４は、改ざん検知機能のみをＳＷ側にソースコードとして組み込み、ビルド部２０２は、ＳＷ側に関するビルドを行う。その後、ビルド部２０２は、残りの全てのソースコードに関して、ＮＷ側のソフトウェアとしてビルドを行う。

【００８５】

第２のパターンは、監視対象のソフトウェアがもともとＳＷを利用する場合である。この場合は、改ざん検知機能組み込み部３０４は、ＳＷ側のソースコードとして新たに改ざん検知機能を組み込み、ビルド部２０２は、もともとＳＷ側に組み込まれていたソースコードと一緒にＳＷ側に関するビルドを行う。その後、ビルド部２０２は、残りの全てのソースコードを、ＮＷ側のソフトウェアとしてビルドする。なお、このようなＴＥＥを用いたビルド方法については、既存のツールを用いて実現できるため詳細な説明は省略する。

【００８６】

［実施の形態３の効果］

次に、本実施の形態３に係る改ざん検知機能組み込み装置３００の効果について説明す

る。
本実施の形態３によれば、改ざん検知機能組み込み部３０４は、ＳＷというハードウェア的に安全性が保障されたメモリ領域に改ざん検知機能を組み込む。その結果、改ざん検知機能に対する攻撃を阻止することができるため、上述した実施の形態１，２よりも、機器の安全な運用を実施することができる。
本実施の形態３は、それ以外の効果は、上述した実施の形態２と同じである。
【００８７】
なお、本実施の形態３は、上述した実施の形態２を変形した例として説明したが、これには限定されない。本実施の形態３は、上述した実施の形態１を変形した例とすることも可能である。
【００８８】
［実施の形態４］
次に、図９を参照して、本実施の形態４に係る改ざん検知機能組み込み装置４００のハードウェア構成例について説明する。
図９を参照すると、改ざん検知機能組み込み装置４００は、プロセッサ４０１と、メモリ４０２と、を含む。
【００８９】
プロセッサ４０１は、例えば、ＭＰＵ（Ｍｉｃｒｏ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）、またはＣＰＵであってもよい。プロセッサ４０１は、複数のプロセッサを含んでもよい。メモリ４０２は、揮発性メモリ及び不揮発性メモリの組み合わせによって構成される。メモリ４０２は、プロセッサ４０１から離れて配置されたストレージを含んでもよい。この場合、プロセッサ４０１は、図示されていないＩ／Ｏインタフェースを介してメモリ４０２にアクセスしてもよい。
【００９０】
上述した実施の形態１，２，３に係る改ざん検知機能組み込み装置１００，２００，３００は、それぞれ、図９に示したハードウェア構成を有することができる。上述した実施の形態１，２，３に係る改ざん検知機能組み込み装置１００，２００，３００の入出力部１０１，２０１、ビルド部１０２，２０２、ＣＦＧ生成部１０３、改ざん検知機能組み込み部１０４，３０４、及びホワイトリスト作成部１０５、２０５は、プロセッサ４０１がメモリ４０２に記憶されたプログラムを読み込んで実行することにより実現されてもよい。また、上述した実施の形態１，２，３に係る改ざん検知機能組み込み装置１００，２００，３００の記憶部１０６は、メモリ４０２により実現されてもよい。
【００９１】
プログラムは、様々なタイプの非一時的なコンピュータ可読媒体（ｎｏｎ－ｔｒａｎｓｉｔｏｒｙ　ｃｏｍｐｕｔｅｒ　ｒｅａｄａｂｌｅ　ｍｅｄｉｕｍ）を用いて格納され、改ざん検知機能組み込み装置１００，２００，３００に供給することができる。非一時的なコンピュータ可読媒体の例は、磁気記録媒体（例えばフレキシブルディスク、磁気テープ、ハードディスクドライブ）、光磁気記録媒体（例えば光磁気ディスク）を含む。さらに、非一時的なコンピュータ可読媒体の例は、ＣＤ－ＲＯＭ（Ｒｅａｄ　Ｏｎｌｙ　Ｍｅｍｏｒｙ）、ＣＤ－Ｒ、ＣＤ－Ｒ／Ｗを含む。さらに、非一時的なコンピュータ可読媒体の例は、半導体メモリを含む。半導体メモリは、例えば、マスクＲＯＭ、ＰＲＯＭ（Ｐｒｏｇｒａｍｍａｂｌｅ　ＲＯＭ）、ＥＰＲＯＭ（Ｅｒａｓａｂｌｅ　ＰＲＯＭ）、フラッシュＲＯＭ、ＲＡＭ（Ｒａｎｄｏｍ　Ａｃｃｅｓｓ　Ｍｅｍｏｒｙ）を含む。また、プログラムは、様々なタイプの一時的なコンピュータ可読媒体（ｔｒａｎｓｉｔｏｒｙ　ｃｏｍｐｕｔｅｒ　ｒｅａｄａｂｌｅ　ｍｅｄｉｕｍ）によって改ざん検知機能組み込み装置１００，２００，３００に供給されてもよい。一時的なコンピュータ可読媒体の例は、電気信号、光信号、及び電磁波を含む。一時的なコンピュータ可読媒体は、電線及び光ファイバ等の有線通信路、又は無線通信路を介して、プログラムを改ざん検知機能組み込み装置１００，２００に供給できる。
【００９２】

以上、実施の形態を参照して本開示を説明したが、本開示は上述した実施の形態に限定されるものではない。本開示の構成や詳細には、本開示のスコープ内で当業者が理解し得る様々な変更をすることができる。

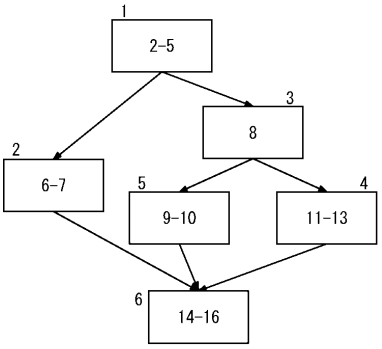
【符号の説明】

【００９３】

１００　改ざん検知機能組み込み装置
１０１　入出力部
１０２　ビルド部
１０３　ＣＦＧ生成部
１０４　改ざん検知機能組み込み部
１０５　ホワイトリスト作成部
１０６　記憶部
２００　改ざん検知機能組み込み装置
２０１　入出力部
２０２　ビルド部
２０５　ホワイトリスト作成部
３００　改ざん検知機能組み込み装置
３０４　改ざん検知機能組み込み部
４００　改ざん検知機能組み込み装置
４０１　プロセッサ
４０２　メモリ

【図面】

【図１】



ソースコード:

```
 1: int func(void){
 2:     int a;
 3:     :
 4:     :
 5:     if(a>1){
 6:         a++;
 7:     }
 8:     else if(a<0){
 9:         a--;
10:     }
11:     else{
12:         a=0;
13:     }
14:     :
15:     :
16: }
```

Fig.1

【図２】

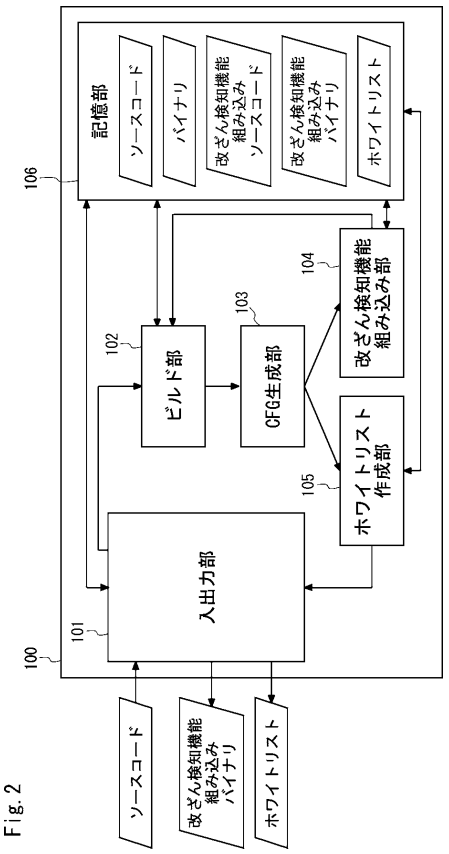


Fig.2

【図３】

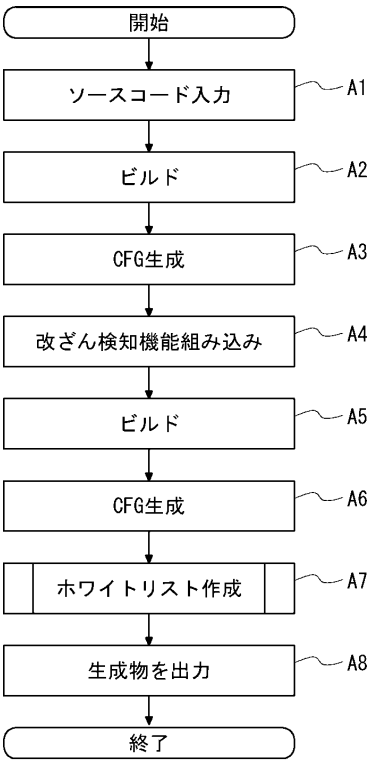


Fig. 3

【図４Ａ】

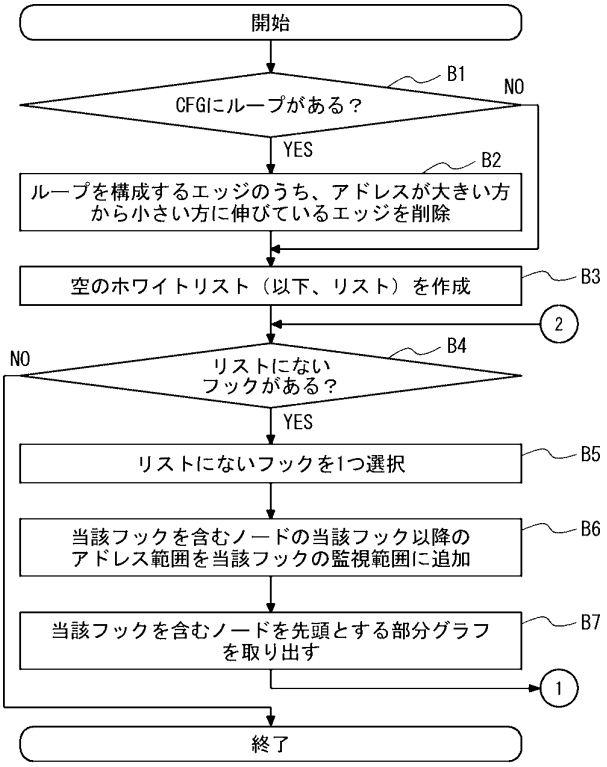


Fig. 4A

【図４Ｂ】

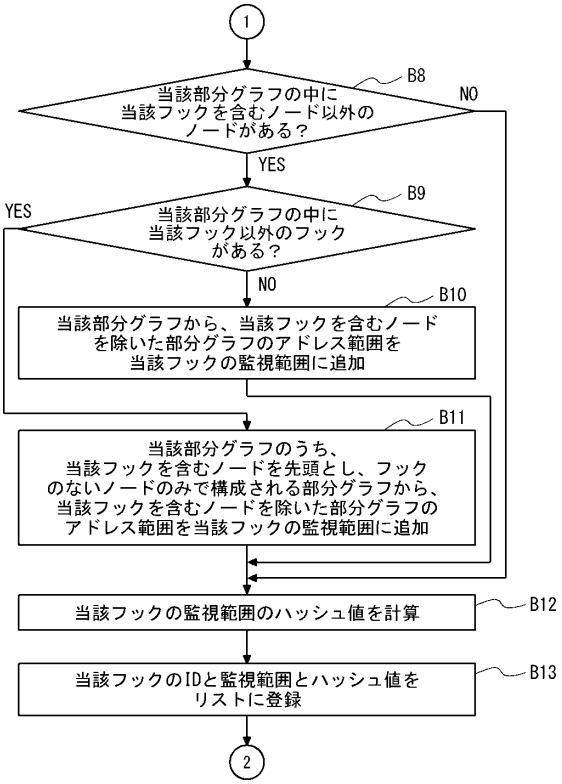


Fig. 4B

【図５】

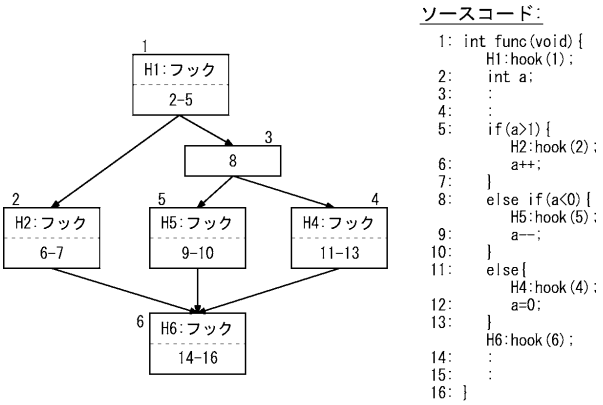


Fig. 5

【図６】

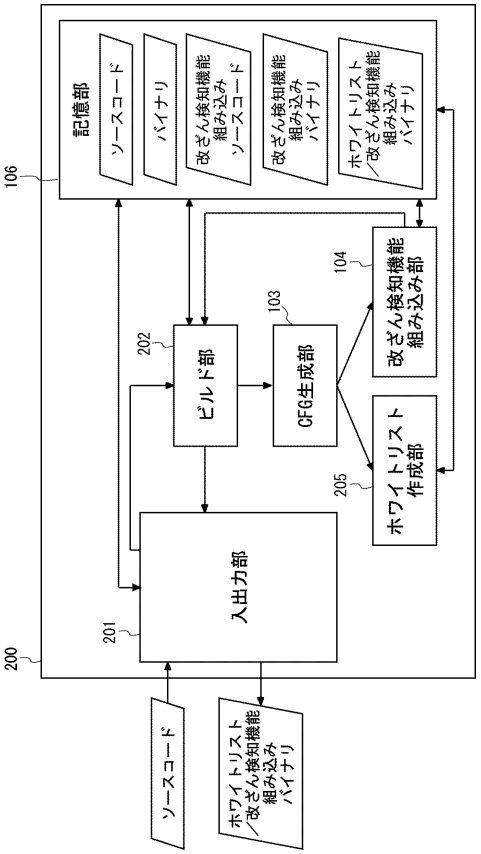


Fig. 6

【図７】

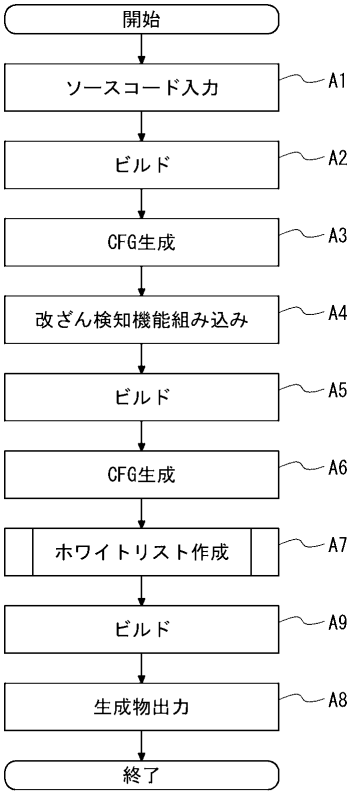


Fig. 7

【図８】

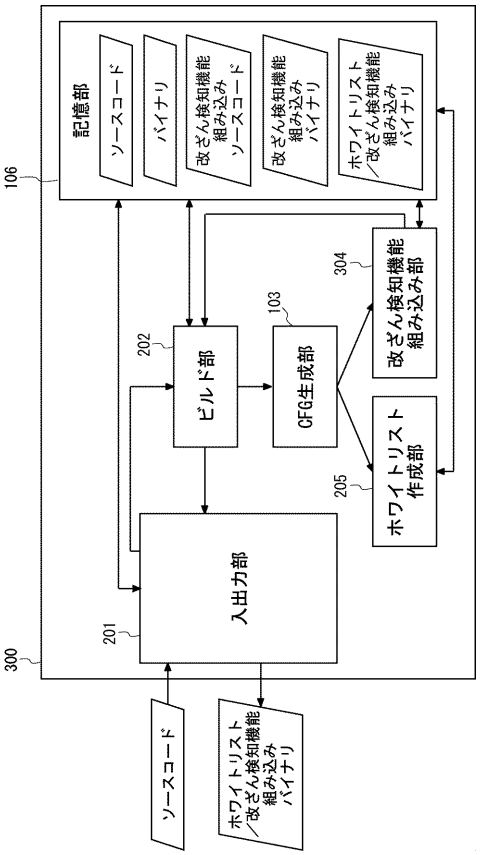


Fig. 8

【図９】

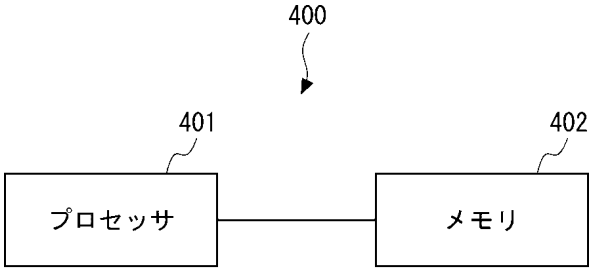


Fig. 9

フロントページの続き

(56)参考文献 国際公開第２０１８／１５０６１９（WO，Ａ１）
国際公開第２０１１／０３３７７３（WO，Ａ１）
ＮＥＣ デジタルプラットフォーム事業部，ＩｏＴデバイスの不正な改ざんを検知する 軽量プログラム改ざん検知 開発キット，Ｃ＆Ｃユーザーフォーラム＆ｉＥＸＰＯ２０１９，日本，ＮＥＣ デジタルプラットフォーム事業部，2019年10月

(58)調査した分野 (Int.Cl.，ＤＢ名)
Ｇ０６Ｆ１２／１４
２１／００－２１／８８
Ｇ０９Ｃ１／００－５／００
Ｈ０４Ｋ１／００－３／００
Ｈ０４Ｌ９／００－９／４０