(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6704503号
(P6704503)
(45)発行日　令和2年6月3日(2020.6.3)　(24)登録日　令和2年5月14日(2020.5.14)

(51)Int.Cl.　F I
G06F 21/56 (2013.01)　G06F 21/56 360

請求項の数 22 (全 37 頁)

(21)出願番号　特願2019-503726(P2019-503726)
(86)(22)出願日　平成29年6月30日(2017.6.30)
(65)公表番号　特表2019-526123(P2019-526123A)
(43)公表日　令和1年9月12日(2019.9.12)
(86)国際出願番号　PCT/US2017/040502
(87)国際公開番号　WO2018/022257
(87)国際公開日　平成30年2月1日(2018.2.1)
審査請求日　令和1年8月28日(2019.8.28)
(31)優先権主張番号　62/368,223
(32)優先日　平成28年7月29日(2016.7.29)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　15/465,515
(32)優先日　平成29年3月21日(2017.3.21)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　507364838
クアルコム，インコーポレイテッド
アメリカ合衆国　カリフォルニア　９２１２１　サン　ディエゴ　モアハウス　ドライブ　５７７５
(74)代理人　100108453
弁理士　村山　靖彦
(74)代理人　100163522
弁理士　黒田　晋平
(72)発明者　サブラト・クマール・デ
アメリカ合衆国・カリフォルニア・９２１２１・サン・ディエゴ・モアハウス・ドライヴ・５７７５

最終頁に続く

(54)【発明の名称】オフセットベースの仮想アドレスマッピングを使用したターゲットアプリケーション機能のカーネルベースの検出

(57)【特許請求の範囲】
【請求項１】
コンピューティングデバイス上で実行されるターゲットアプリケーションの機能を検出するための方法であって、
コンピューティングデバイス上のセキュアなメモリに、アプリケーションのためのアプリケーション固有仮想アドレスマッピングテーブルを記憶するステップであって、前記アプリケーション固有仮想アドレスマッピングテーブルは、前記アプリケーションのソースコードにおける対応するターゲットアプリケーションの機能にマッピングされたアプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含む、ステップと、
前記アプリケーションを起動したことに応答して、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成するステップであって、前記プロセス固有仮想アドレスマッピングテーブルは、前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して、前記ターゲットアプリケーションの機能に対応する実際の仮想アドレスを定義する、ステップと、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行中に、前記プロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数におけるバイナリ命令が実行されることを検出するステップと、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイ

ナリコードの実行に関連付けられた1つまたは複数の悪意のあるコードまたは挙動を検出するように構成された検出アルゴリズムに、前記プロセス固有仮想アドレスマッピングテーブルにおける前記実際の仮想アドレスから検出された、前記実行されたターゲットアプリケーションの機能に関する情報を提供するステップと
を含む方法。
【請求項２】
前記アプリケーションプロセスの他のインスタンスと同時に実行される前記アプリケーションプロセスの別のインスタンスのために別のプロセス固有仮想アドレスマッピングテーブルを生成するステップと、
前記アプリケーションプロセスの前記別のインスタンスのための前記アプリケーションバイナリコードの実行中に、前記別のプロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数における前記バイナリ命令がいつ実行されるかを検出するステップとをさらに含む、請求項1に記載の方法。
【請求項３】
前記実際の仮想アドレスは、前記アプリケーションプロセスの前記インスタンスのベース仮想アドレスおよび前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して決定される、請求項1に記載の方法。
【請求項４】
前記セキュアなメモリは、ハイレベルオペレーティングシステム(HLOS)における信頼できるゾーンに存在する、請求項1に記載の方法。
【請求項５】
前記アプリケーションは、セキュアなウェブアプリケーションおよびウェブブラウザのうちの1つを含む、請求項1に記載の方法。
【請求項６】
前記アプリケーションバイナリコードは、ネイティブバイナリコードとして実行される、請求項1に記載の方法。
【請求項７】
コンピューティングデバイス上で実行されるターゲットアプリケーションの機能を検出するためのシステムであって、
コンピューティングデバイス上に、アプリケーションのためのアプリケーション固有仮想アドレスマッピングテーブルを記憶するための手段であって、前記アプリケーション固有仮想アドレスマッピングテーブルは、前記アプリケーションのソースコードにおける対応するターゲットアプリケーションの機能にマッピングされたアプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含む、手段と、
前記アプリケーションを起動したことに応答して、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成するための手段であって、前記プロセス固有仮想アドレスマッピングテーブルは、前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して、前記ターゲットアプリケーションの機能に対応する実際の仮想アドレスを定義する、手段と、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行中に、前記プロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数におけるバイナリ命令が実行されることを検出するための手段と、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行に関連付けられた1つまたは複数の悪意のあるコードまたは挙動を検出するように構成された検出アルゴリズムに、前記プロセス固有仮想アドレスマッピングテーブルにおける前記実際の仮想アドレスから検出された、前記実行されたターゲットアプリケーションの機能に関する情報を提供するための手段と

を含むシステム。
【請求項８】
前記アプリケーションプロセスの他のインスタンスと同時に実行される前記アプリケーションプロセスの別のインスタンスのために別のプロセス固有仮想アドレスマッピングテーブルを生成するための手段と、
前記アプリケーションプロセスの前記別のインスタンスのための前記アプリケーションバイナリコードの実行中に、前記別のプロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数における前記バイナリ命令がいつ実行されるかを検出するための手段と
をさらに含む、請求項7に記載のシステム。
【請求項９】
前記実際の仮想アドレスは、前記アプリケーションプロセスの前記インスタンスのベース仮想アドレスおよび前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して決定される、請求項7に記載のシステム。
【請求項１０】
記憶するための前記手段は、ハイレベルオペレーティングシステム(HLOS)における信頼できるゾーンに存在する、請求項7に記載のシステム。
【請求項１１】
前記アプリケーションは、セキュアなウェブアプリケーションおよびウェブブラウザのうちの1つを含む、請求項7に記載のシステム。
【請求項１２】
前記アプリケーションバイナリコードは、ネイティブバイナリコードとして実行される、請求項7に記載のシステム。
【請求項１３】
メモリ内で具現化され、コンピューティングデバイス上で実行されるターゲットアプリケーションの機能を検出するためにプロセッサによって実行可能であるコンピュータ可読プログラムコードを有するコンピュータプログラムであって、
コンピューティングデバイス上のセキュアなメモリに、アプリケーションのためのアプリケーション固有仮想アドレスマッピングテーブルを記憶することであって、前記アプリケーション固有仮想アドレスマッピングテーブルは、前記アプリケーションのソースコードにおける対応するターゲットアプリケーションの機能にマッピングされたアプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含む、記憶することと、
前記アプリケーションを起動したことに応答して、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成することであって、前記プロセス固有仮想アドレスマッピングテーブルは、前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して、前記ターゲットアプリケーションの機能に対応する実際の仮想アドレスを定義する、生成することと、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行中に、前記プロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数におけるバイナリ命令が実行されることを検出することと、
前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行に関連付けられた1つまたは複数の悪意のあるコードまたは挙動を検出するように構成された検出アルゴリズムに、前記プロセス固有仮想アドレスマッピングテーブルにおける前記実際の仮想アドレスから検出された、前記実行されたターゲットアプリケーションの機能に関する情報を提供することと
を行うように構成された論理を含むコンピュータプログラム。
【請求項１４】

前記アプリケーションプロセスの他のインスタンスと同時に実行される前記アプリケーションプロセスの別のインスタンスのために別のプロセス固有仮想アドレスマッピングテーブルを生成することと、
前記アプリケーションプロセスの両方のインスタンスのための前記アプリケーションバイナリコードの同時実行中に、前記別のプロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数における前記バイナリ命令がいつ実行されるかを検出することと
を行うように構成された論理をさらに含む、請求項13に記載のコンピュータプログラム。
【請求項１５】
前記実際の仮想アドレスは、前記アプリケーションプロセスの前記インスタンスのベース仮想アドレスおよび前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して決定される、請求項13に記載のコンピュータプログラム。
【請求項１６】
前記セキュアなメモリは、ハイレベルオペレーティングシステム(HLOS)における信頼できるゾーンに存在する、請求項13に記載のコンピュータプログラム。
【請求項１７】
前記アプリケーションは、セキュアなウェブアプリケーションおよびウェブブラウザのうちの1つを含む、請求項13に記載のコンピュータプログラム。
【請求項１８】
前記アプリケーションバイナリコードは、ネイティブバイナリコードとして実行される、請求項13に記載のコンピュータプログラム。
【請求項１９】
実行中のターゲットアプリケーションの機能を検出するためのシステムであって、
アプリケーションバイナリコードを実行するように構成された処理デバイスと、
ハイレベルオペレーティングシステム(HLOS)と
を含み、前記HLOSは、
アプリケーションのソースコードにおける対応するターゲットアプリケーションの機能にマッピングされた前記アプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含むアプリケーション固有仮想アドレスマッピングテーブルと、
前記アプリケーションを起動したことに応答して、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成するように構成されたカーネルモジュールであって、前記プロセス固有仮想アドレスマッピングテーブルは、前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して、前記ターゲットアプリケーションの機能に対応する実際の仮想アドレスを定義する、カーネルモジュールと
を含み、
前記HLOSは、前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行中に、前記プロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数におけるバイナリ命令が実行されることを検出するように構成され、
前記HLOSは、前記アプリケーションプロセスの前記インスタンスのための前記アプリケーションバイナリコードの実行に関連付けられた1つまたは複数の悪意のあるコードまたは挙動を検出するように構成された検出アルゴリズムに、前記プロセス固有仮想アドレスマッピングテーブルにおける前記実際の仮想アドレスから検出された、前記実行されたターゲットアプリケーションの機能に関する情報を提供する、システム。
【請求項２０】
前記HLOSは、
前記アプリケーションプロセスの他のインスタンスと同時に実行される前記アプリケー

ションプロセスの別のインスタンスのために別のプロセス固有仮想アドレスマッピングテーブルを生成することと、
前記アプリケーションプロセスの両方のインスタンスのための前記アプリケーションバイナリコードの同時実行中に、前記別のプロセス固有仮想アドレスマッピングテーブルに基づいて、前記ターゲットアプリケーションの機能に対応する前記実際の仮想アドレスのうちの1つまたは複数における前記バイナリ命令がいつ実行されるかを検出することとを行うようにさらに構成される、請求項19に記載のシステム。
【請求項２１】
前記実際の仮想アドレスは、前記アプリケーションプロセスの前記インスタンスのベース仮想アドレスおよび前記アプリケーション固有仮想アドレスマッピングテーブルにおける前記仮想アドレスオフセットを使用して決定される、請求項20に記載のシステム。
【請求項２２】
前記アプリケーション固有仮想アドレスマッピングテーブルは、前記HLOSにおける信頼できるゾーンに記憶される、請求項20に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
優先権および関連出願の陳述
本出願は、その全体が参照により本明細書に組み込まれる、2016年7月29日に出願され「Kernel-Based Detection of Target Application Functionality Using Virtual Address Mapping」と題する米国仮特許出願第62/368,223号(Qualcomm整理番号163161P1)に対する米国特許法第119条(e)項に基づく優先権を主張する。
【０００２】
本出願はまた、2016年8月23日に出願され「Kernel-Based Detection of Target Application Functionality Using Virtual Address Mapping」と題する米国特許出願第15/245,037号(整理番号163161U1)、および2016年8月23日に出願され「Updating Virtual Memory Addresses of Target Application Functionalities for an Updated Version of Application Binary Code」と題する米国特許出願第15/245,041号(整理番号163161U2)に関する。
【背景技術】
【０００３】
システムまたはプラットフォームレイヤにおける顕著な活動を示さず、したがってアプリケーション実行の有用な機能情報および挙動情報を検出する機会を提供しないハードウェアプラットフォーム上で実行される様々なハイレベルアプリケーションがある。損なわれたハイレベルウェブブラウザアプリケーションである一般的な例。
【０００４】
システムまたはプラットフォームレイヤにおける顕著な活動を示さず、したがってアプリケーション実行の有用な機能情報および挙動情報を検出する機会を提供しないハードウェアプラットフォーム上で実行される様々なハイレベルアプリケーションがある。システムおよびプラットフォームレベルにおいて痕跡(indicative trace)を残さないデバイス上での実行中のセキュリティエクスプロイト(たとえば、クロスサイトスクリプティング)に対して損なわれたハイレベルウェブブラウザアプリケーションである一般的な例。システムライブラリ、プラットフォーム、SOCハードウェアを精査すること、またはデバイスレベル活動を注視することのいずれかによって、ハイレベルアプリケーション上でそのような活動が発生していると判断する方法がない。したがって、デバイス上で実行される様々なサードパーティ製アプリケーションに対するプラットフォームレベル制御を改善するために、またこれらの実行中のハイレベルアプリケーションの機能活動および挙動活動の一部を検出するために、プラットフォームのHLOSまたはカーネルが理解できる形式でハイレベルアプリケーションの機能および挙動を表現および通信することを可能にする機構を開発する必要がある。これにより、プラットフォームが実行中のアプリケーションの挙動に

対する理解を深めることができるようになり、プラットフォームが実行中のアプリケーションの様々な異なる状況に対処するために決定し、措置を講じることができるようになる。一例として、サードパーティ製ウェブブラウザアプリケーション上のウェブセキュリティエクスプロイトを防止するプラットフォームレベルの決定が、情報を使用して行われ得る。例示的な使用の他の領域は、HLOSまたはカーネルレイヤにおいて本開示における機構を使用してアプリケーションの特定の機能的性質または挙動性質が検出されると、プラットフォームが様々なSOC構成要素(DDR、バス、CPU、キャッシュ)の周波数を引き上げる/引き下げるなどの決定を行うことであり、または高電力モードもしくは低電力モードを関与させる。一般に、本開示により、プラットフォームは、デバイス上で実行される様々なサードパーティ製アプリケーションに対する様々な制御を、アプリケーションによって実行されている機能を検出および認識することによって行う機会を得る。これにより、SOCおよびプラットフォームベンダーは、プラットフォームが本来制御できない様々なサードパーティ製アプリケーションに対して、プラットフォームレベルからのより良いソリューションを提供することができるようになる。

【発明の概要】

【課題を解決するための手段】

【０００５】

コンピューティングデバイス上で実行されるアプリケーションのハイレベル機能を検出するためのシステム、方法、およびコンピュータプログラムが開示される。方法の一実施形態は、コンピューティングデバイス上のセキュアなメモリに、アプリケーションのためのアプリケーション固有仮想アドレスマッピングテーブルを記憶するステップを含む。アプリケーション固有仮想アドレスマッピングテーブルは、対応するターゲットアプリケーション機能にマッピングされたアプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含む。アプリケーションを起動したことに応答して、本方法は、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成する。プロセス固有仮想アドレスマッピングテーブルは、アプリケーション固有仮想アドレスマッピングテーブルにおける仮想アドレスオフセットを使用して、ターゲットアプリケーション機能に対応する実際の仮想アドレスを定義する。アプリケーションプロセスのインスタンスのためのアプリケーションバイナリコードの実行中に、本方法は、プロセス固有仮想アドレスマッピングテーブルに基づいて、ターゲットアプリケーション機能に対応する実際の仮想アドレスのうちの1つまたは複数がいつ実行されるかを検出する。

【０００６】

別の実施形態は、アプリケーションバイナリコードを実行するように構成された処理デバイスとハイレベルオペレーティングシステム(HLOS)とを含むシステムである。HLOSは、対応するターゲットアプリケーション機能にマッピングされたアプリケーションバイナリコードにおける複数の仮想アドレスオフセットを含むアプリケーション固有仮想アドレスマッピングテーブルを含む。HLOSは、アプリケーションを起動したことに応答して、実行されるアプリケーションプロセスのインスタンスのためにプロセス固有仮想アドレスマッピングテーブルを生成するように構成されたカーネルモジュールをさらに含む。プロセス固有仮想アドレスマッピングテーブルは、アプリケーション固有仮想アドレスマッピングテーブルにおける仮想アドレスオフセットを使用して、ターゲットアプリケーション機能に対応する実際の仮想アドレスを定義する。HLOSは、アプリケーションプロセスのインスタンスのためのアプリケーションバイナリコードの実行中に、プロセス固有仮想アドレスマッピングテーブルに基づいて、ターゲットアプリケーション機能に対応する実際の仮想アドレスのうちの1つまたは複数がいつ実行されるかを検出するように構成される。

【０００７】

図では、同様の参照番号は、その他の形で示されない限り、様々な図の全体を通して同様の部分を指す。「102A」または「102B」などの文字指定を伴う参照番号の場合、文字指定は、同じ図に存在する2つの同様の部分または要素を区別することができる。参照番号

がすべての図において同じ参照番号を有するすべての部分を含むことが意図されるとき、参照番号に対する文字指定は省略される場合がある。

【図面の簡単な説明】

【０００８】

【図１】セキュアなメモリにおける仮想アドレスマッピングを使用してターゲットアプリケーション機能を検出するためのシステムの実施形態のブロック図である。

【図２】対応するアプリケーションバイナリコードへのターゲットアプリケーション機能の例示的なマッピングを示す図である。

【図３】仮想アドレス-関数マッピングテーブル(VAFMT)の例示的な実施形態を示す図である。

【図４】図1のシステムにおける悪意のあるコード活動を検出するための方法の実施形態を示すフローチャートである。

【図５】仮想機械コード空間の境界を動的に識別するために使用されるVAFMTの別の実施形態を示す図である。

【図６】VAFMTと組み合わせて使用される識別子-仮想マッピングテーブル(IVAMT)の実施形態を示す図である。

【図７】ガベージコレクションプロセスに関連して使用されるVMコード空間の一部分を示す図である。

【図８】図1の仮想機械におけるガベージコレクション関数の例示的な注目ポイント、および仮想機械を含むアプリケーションバイナリの実行中にガベージコレクション活動の実行を検出するために使用されるVAFMTにおける機能注目ポイントの仮想アドレスを示す図である。

【図９】仮想機械ヒープの外部/内部境界の仮想アドレスの例示的なマッピングを示す図である。

【図１０】仮想機械実施形態において図1のシステムにおける悪意のあるコード活動を検出するための方法の実施形態を示すフローチャートである。

【図１１】特定のデータ構造タイプのオブジェクトを含む動的に割り振られたバッファの仮想アドレスを決定するために使用される特定のバッファアロケータ関数の仮想アドレス、およびバッファにおいて割り振られたオブジェクトのメンバー/フィールドの値を含むVAFMTの実施形態を示す図である。

【図１２】アプリケーションバイナリコードの更新バージョンを受信したことに応答してVAFMTを自動的に更新するためのシステムの実施形態を示す合成ブロック/フロー図である。

【図１３】更新された仮想アドレスおよびメタデータとともに図12のVAFMTを示す図である。

【図１４】疑似バイナリコードテンプレートへの図12のVAFMTにおける機能注目ポイントの例示的なマッチングを示す図である。

【図１５】アプリケーションバイナリコードの更新バージョンにおけるマッチング領域への図14の疑似バイナリコードテンプレートの例示的なマッチングを示す図である。

【図１６】アプリケーションバイナリコードの更新バージョンを受信したことに応答してVAFMTを更新するための方法の実施形態を示すフローチャートである。

【図１７】オフセットベースの仮想アドレスマッピングを使用してターゲットアプリケーション機能を検出するためのシステムの実施形態のブロック/フロー図である。

【図１８】図17におけるアプリケーション固有VAFMTの例示的な実施形態を示す図である。

【図１９】図17におけるプロセス固有VAFMTのうちの1つの例示的な実施形態を示す図である。

【図２０】アプリケーション固有URLバッファVAFMTの別の実施形態を示す図である。

【図２１】図20のアプリケーション固有URLバッファVAFMTにおいて識別される第1のアプリケーションのためのプロセス固有VAFMTの実施形態を示す図である。

【図２２】図20のアプリケーション固有URLバッファVAFMTにおいて識別される第2のアプリケーションのためのプロセス固有VAFMTの実施形態を示す図である。
【図２３】オフセットベースの仮想アドレスマッピングを使用してターゲットアプリケーション機能を検出するための方法の実施形態を示すフローチャートである。
【発明を実施するための形態】
【０００９】
「例示的」という語は、本明細書では「例、事例、または例示として機能すること」を意味するために使用される。本明細書で「例示的」と記載されている任意の態様は、必ずしも他の態様よりも好ましいまたは有利であると解釈されるべきではない。
【００１０】
本明細書では、「アプリケーション」という用語は、オブジェクトコード、スクリプト、バイトコード、マークアップ言語ファイル、およびパッチなどの、実行可能コンテンツを有するファイルも含み得る。加えて、本明細書で言及する「アプリケーション」は、開かれる必要があり得るドキュメント、またはアクセスされる必要がある他のデータファイルなどの、本質的に実行可能ではないファイルも含み得る。
【００１１】
「コンテンツ」という用語は、オブジェクトコード、スクリプト、バイトコード、マークアップ言語ファイル、およびパッチなどの、実行可能コンテンツを有するファイルも含み得る。加えて、本明細書で言及する「コンテンツ」は、開かれる必要があり得るドキュメント、またはアクセスされる必要がある他のデータファイルなどの、本質的に実行可能ではないファイルも含み得る。
【００１２】
本明細書で使用する「構成要素」、「データベース」、「モジュール」、「システム」などの用語は、ハードウェア、ファームウェア、ハードウェアとソフトウェアの組合せ、ソフトウェア、または実行中のソフトウェアのいずれかのコンピュータ関連エンティティを指すものとする。たとえば、構成要素は、限定はしないが、プロセッサ上で実行されるプロセス、プロセッサ、オブジェクト、実行ファイル、実行のスレッド、プログラム、および/またはコンピュータであってもよい。例として、コンピューティングデバイス上で実行されるアプリケーションとコンピューティングデバイスの両方が構成要素であってもよい。1つまたは複数の構成要素は、プロセスおよび/または実行のスレッド内に存在してもよく、構成要素は、1つのコンピュータ上に局在化されてもよく、かつ/または2つ以上のコンピュータ間に分散されてもよい。さらに、これらの構成要素は、様々なデータ構造が記憶された様々なコンピュータ可読媒体から実行することができる。構成要素は、1つまたは複数のデータパケット(たとえば、ローカルシステム、分散システムの中の別の構成要素と、かつ/またはインターネットなどのネットワークにわたって信号によって他のシステムと対話する1つの構成要素からのデータなど)を有する信号に従うなどして、ローカルプロセスおよび/またはリモートプロセスによって通信することができる。
【００１３】
図1は、カーネルまたはオペレーティングシステム(O/S)レイヤからのアプリケーションバイナリの所望のまたはターゲットハイレベル機能を検出するためのシステム100の実施形態を示す。図1の実施形態に示すように、システム100は、処理デバイス(たとえば、中央処理装置(CPU)102)、メモリ104、およびハイレベルオペレーティングシステム(HLOS)106を含む。メモリ104は、CPU102によって実行され得る1つまたは複数のアプリケーションを記憶する。メモリ104は、コンピューティングデバイス上にインストールされたアプリケーションに関連付けられた参照アプリケーションソースコード110に対応するアプリケーションバイナリコード108を記憶し得る。この点について、システム100は、たとえば、パーソナルコンピュータ、ラップトップコンピュータ、ワークステーション、サーバ、または、セルラー電話、スマートフォン、携帯情報端末(PDA)、ポータブルゲームコンソール、ナビゲーションデバイス、タブレットコンピュータ、ウェアラブルデバイス(たとえば、スマートウォッチ)、もしくは他のバッテリー電源式ポータブルデバイスなどのポー

タブルコンピューティングデバイス(PCD)を含む、任意の所望のコンピューティングデバイスまたはシステムにおいて実装され得る。

【００１４】

一実施形態では、カーネルまたはO/Sレイヤは、ハイレベルオペレーティングシステム(HLOS)106を含む。図1に示すように、HLOS106は、登録アプリケーションのリスト112、セキュアなメモリ(たとえば、信頼できるゾーン114)、および各登録アプリケーションのアプリケーションバイナリコード108のための特別に構成された仮想アドレスマッピングテーブルを含む。登録アプリケーションのリスト112は、セキュアな制御および/またはサポートのためにHLOS106に登録されているシステム100上にインストールされたアプリケーションを識別する。たとえば、アプリケーション(たとえば、ウェブアプリケーション、ブラウザアプリケーションなど)のアプリケーションバイナリコード108は、HLOS106に登録され、リスト112において識別され得る。当技術分野で知られているように、信頼できるゾーン114は、メモリにロードされ、かつ/または実行されるコードおよび/またはデータがセキュリティ、秘密性、完全性などに関して保護されることを保証するように構成されたセキュアなメモリまたはエリアを含む。登録アプリケーションのためのアプリケーションバイナリコード108は、所定の仮想アドレスポイントの実行を追跡することによって所望のまたはターゲットハイレベルアプリケーション機能を識別するためにHLOS106および/または信頼できるゾーン114におけるアルゴリズムによって使用される、1つまたは複数の仮想アドレスマッピングテーブルを有し得る。

【００１５】

システム100は、カーネルレイヤにおけるハイレベルアプリケーション機能の追跡および検出が有利である様々なアプリケーションドメインに適用され得ることを諒解されたい。たとえば、例示的な一実施形態では、カーネルは、実行中のアプリケーションの特定の機能的性質または挙動性質の検出に応答して、様々なシステムオンチップ(SoC)構成要素(たとえば、中央処理装置(CPU)、キャッシュ、ダブルデータレート(DDR)メモリ、1つもしくは複数のバスなど)の周波数を引き上げることおよび/もしくは引き下げることなど、決定を制御すること、または高電力モードおよび/もしくは低電力モードを設定すること、ならびに特定のハードウェア特徴を有効化/無効化することができる。このようにして、HLOS106およびカーネルは、デバイス上で実行される様々なサードパーティ製アプリケーションに対する様々な制御を、アプリケーションによって実行されている機能を検出および認識することによって実施する機会を有する。これにより、SOCおよびプラットフォームベンダーは、プラットフォームが本来制御できないことがある様々なサードパーティ製アプリケーションに対して、プラットフォーム/HLOS/カーネルレベルからの改善されたソリューションを提供することができるようになることを諒解されたい。

【００１６】

例示的なアプリケーションドメインでは、システム100は、ウェブアプリケーション、ウェブブラウザ、JavaScriptコード(「JavaScript」は登録商標。以下同じ。)などの悪意のある攻撃または他のエクスプロイトに対するリアルタイムのセキュリティ保護を実現し得る。当技術分野で知られているように、JavaScriptは、多くのウェブサイトおよびウェブアプリケーションにおいて使用されるプログラミング言語であり、JavaScriptベースの攻撃は、サイバーセキュリティに対する最上位の脅威のうちの1つである。ますます多くのウェブ活動がデスクトップコンピュータからモバイルにシフトするにつれて、JavaScript攻撃は、ポータブルコンピューティングデバイスに対する主要な脅威になりつつある。

【００１７】

たいていの悪意のあるJavaScript攻撃は、エクスプロイトのためにJavaScript言語の特性ならびにウェブ標準および仕様の制約を利用する。悪意のあるJavaScriptを通じたウェブベースのエクスプロイトの一般的な例には、クロスサイトスクリプティング(すなわち、XSS/CSS)、クロスサイトリクエストフォージェリ(すなわち、CSRF/XSRF)、ドライブバイダウンロード、ユーザ意図ハイジャッキング(user intent hijacking)、クリックジャッキング、分散型サービス拒否(DDoS)、JavaScriptステガノグラフィ、および様々な形式

の難読化されたJavaScriptが含まれる。悪意のある挙動を検出しようと試みる際には、ハイレベルのウェブの挙動および機能の知識が必要とされるので、通常、最新のウェブおよびJavaScriptのセキュリティソリューションがブラウザソフトウェアアーキテクチャ内に組み込まれている。

【００１８】

しかしながら、ウェブ/JavaScriptベースのエクスプロイトは、プラットフォーム活動(たとえば、システムコール、デバイス使用など)に対する可視の指示を有しないことがあるので、HLOS、カーネルおよびデバイスプラットフォーム内のインビルトウェブセキュリティ機構は限られている。多くのウェブ/JavaScriptベースの攻撃は、外向きであり、ユーザのオンライン資産、活動、識別情報などを損なうだけである。言い換えれば、可視の活動パターンは、ウェブブラウザ/アプリケーションソフトウェア内で検出されるだけであり得、したがって、ウェブエクスプロイトに対する大半のセキュリティ機構が、ほとんど常に、ウェブブラウザアプリケーション内に組み込まれている。

【００１９】

この点について、システム100におけるアプリケーションバイナリコード108の例示的な実施形態は、ウェブアプリケーション、ブラウザアプリケーション、またはHLOS106が所定の仮想アドレスポイントを追跡することによってハイレベルアプリケーション機能を検出する他のアプリケーションを含み得る。図1にさらに示すように、システム100は、信頼できるゾーン114に存在する1つまたは複数の悪意のあるコード検出アルゴリズム116をさらに含み得る。悪意のあるコード検出アルゴリズム116は、仮想アドレスポイントの実行および仮想アドレスマッピングテーブルにおいて識別されるそれらの関連する機能的意味に関係するデータを受信し得る。このデータに基づいて、アルゴリズム116は、たとえば、悪意のあるコードおよび挙動、悪意のあるJavaScriptコードおよび実行などを検出し、セキュリティ脅威を解決するか、またはさもなければ悪意のある攻撃を妨害するための適切な方法を開始することができる。一実施形態では、セキュリティ脅威が検出されたとき、システム100は、脅威を自動的に解決するか、または適切な措置を講じるようユーザに指示し得る。

【００２０】

図1の実施形態に示すように、HLOS106によって使用される仮想アドレスマッピングテーブルは、仮想アドレス-関数マッピングテーブル120および識別子-仮想アドレスマッピングテーブル122を含み得る。HLOS106ならびにマッピングテーブル120および122は、システム100が実行中のアプリケーションバイナリコード108からの所望のまたはターゲットハイレベル機能情報を決定し得る統合プラットフォーム機構を含むことを諒解されたい。ハイレベル機能情報は、悪意のある挙動を検出するために信頼できるゾーン114において実装されたアルゴリズムおよび/またはモデル(たとえば、悪意のあるコード検出アルゴリズム116)によって使用され得る。

【００２１】

以下でより詳細に説明するように、システム100は、アプリケーションバイナリコード108を実行するための2つの異なる実行モデルをサポートし得る。第1の実行モデルは、(たとえば、C/C++コードからの)ネイティブバイナリ実行を伴う。第2の実行モデルは、管理されたランタイム実行(たとえば、仮想機械118による実行)を伴う。一実施形態では、仮想機械118は、JavaScriptソースからの動的ジャストインタイム(JIT)または解釈されたコードを実行し得る。管理されたランタイム実行実施形態では、仮想機械118は、バイナリコード108内で仮想機械118が実行するバイナリコード108の一部を含み得る。しかしながら、他の実施形態では、別個のVMおよびバイナリ作業負荷があり得ることを諒解されたい。

【００２２】

ネイティブバイナリ実行モデルの例示的な実施形態が図2～図4に示されている。ネイティブバイナリ実行のために、登録アプリケーションのリスト112における各アプリケーションは、HLOS106によって維持される対応するVAFMT120を有する。VAFMT120は、信頼でき

るゾーン114に存在し得る。VAFMT120は、関連ハイレベル機能とマッピングされた異なる注目仮想アドレスを含む。一実施形態では、各関連ハイレベル機能は、アルゴリズム116が理解するマクロ名として示され得る。しかしながら、たとえば、特定の仮想アドレスにおいて検出された活動が、アルゴリズム116においてトリガされる必要がある機能に直接対応するような、アルゴリズム116における関数または関数名へのポインタを含む、関連ハイレベル機能を表すための他の機構が実装されてよいことを諒解されたい。バイナリ画像における特定のアプリケーション関数(および関数内の特定のポイント)の仮想アドレスは、「注目ポイント」と呼ばれることがある。一実施形態では、仮想アドレス注目ポイントは、たとえば、機密ソース/シンクルーチン、危険なウェブアプリケーションプログラムインターフェース(API)、特定のウェブ機能、バッファの開始/終了、または攻撃者が利用し得る任意の他のオブジェクトもしくは既知のウェブ/JavaScript攻撃の分析および検出のための他の適切な情報の中、始めもしくは終わりのポイントを含み、またはそれらの間の複数の特定のポイントを含むことがある。他の実施形態では、仮想アドレス注目ポイントは、JavaScriptインタープリタ、ジャストインタイム(JIT)コンパイラ、またはランタイム環境(たとえば、JavaScriptソースコード、バイトコード/JITコードなどを記憶する仮想機械ヒープのための割振り/割振り解除関数)の実装形態におけるポイントを含む。

【００２３】

図2および図3は、VAFMT120の例示的な実施形態を示す。図2は、アプリケーションバイナリコード108内の対応する仮想アドレスポイントへのアプリケーションソースコード110内のいくつかの所望のまたはターゲット機能ポイントの論理マッピング200を示す。図2および図3では、仮想アドレスが示されているが、バイナリオブジェクトコードは示されていない。この実施形態では、アプリケーションソースコード110は、「documentWrite」関数のためのC++コードを含む。ソースコードにおけるポイント201は、バイナリコードにおける仮想アドレス202にマッピングされる。ソースコードにおけるポイント203は、バイナリコードにおける仮想アドレス204にマッピングされる。ソースコードのポイント205は、バイナリコードにおける仮想アドレス206にマッピングされる。図3は、VAFMT120における列302にあるバイナリコードにおける仮想アドレス202、204、および206の、それらの仮想アドレスにおけるコードが表すそれぞれの機能的意味への論理マッピング300を示す。図3に示すように、VAFMT120は、機能注目ポイント(列304)の対応する説明を伴う複数の仮想アドレス(列302)を含み得る。バイナリコードポイントに関する202によって表される仮想アドレス(0x3273fac8)は、DOCUMENT_WRITE_FUNCTION_STARTに対応する機能ポイントにマッピングされる。DOCUMENT_WRITE_1を示す機能注目ポイントに対応するバイナリコードポイントに関する204によって表される仮想アドレス(0x3473fad4)。バイナリコードにおける206によって表される仮想アドレス(0x3473fae8)は、マクロ的意味DOCUMENT_WRITE_2を有する機能ポイントにマッピングされる。

【００２４】

図11は、特定のデータ構造タイプ(たとえば、クラス、構造、集合)のオブジェクトを含む動的に割り振られたバッファの開始および終了の仮想アドレスを決定するために使用され得る特定のバッファアロケータ関数の仮想アドレスを有するカスタム仮想アドレステーブルを含むVAFMT120の実施形態を示す。バッファにおいて割り振られたオブジェクトのメンバー/フィールドの値は、注目ポイントである特定のフィールド/メンバーのためのテーブルにおいて維持されることもある、オフセットおよび長さフィールドを使用して決定され得る。たとえば、システムメモリアロケータ関数の実行をアロケータ関数の仮想アドレスによってカバーされる領域から追跡することによって、割り振られたバッファのサイズおよびアドレスを検出するために、バッファ割振り関数の仮想アドレスが使用され得る。バッファ開始および終了仮想アドレスが知られると、特定のデータ構造タイプに関するオブジェクトの特定のメンバー/フィールドの値を決定するために、オフセットおよび長さフィールドが使用され得る。

【００２５】

図1において破線によって示されているように、アプリケーションソースコード110は、

システム100に記憶される必要がない。そうではなく、それは、オフラインまたはオフデバイスに位置してよく、参照またはオープンソースコードとして利用可能であってよい。ブラウザまたはウェブアプリケーションの実際の商用バイナリにおける注目仮想アドレスを決定するために、参照および指針として特定のバージョンの参照ソースコードが使用され得る。オープンソースプロジェクトのマッチングコード改訂/バージョンから、同等のバイナリがコンパイルされ得る。そのバージョン/改訂に基づくアプリケーションバイナリの所望のまたはターゲット仮想アドレスおよび関数/ポイントを検出するために、参照として、コンパイルされたバイナリが使用され得る。同様のコンパイラおよびリンカオプションが使用されてよい。さらに、アプリケーションコードにおける様々なポイントにおけるブレークポイントが、仮想アドレスおよびそれらの機能マッピングポイントの決定に使用され得る。オープンソースプロジェクトのための既知のコンパイルされた関数からの参照バイナリを使用することによって、所与のアプリケーションバイナリにおける機能を識別するために、バイナリコード認識および類似性抽出方法が利用され得る。わずかに修正されたバージョンを有するバイナリ(または既知の参照オープンソースプロジェクトとのいくつかのソースコード差異を有するソースベースから生じたバイナリ)の場合、重要なウェブ関数およびAPIを呼び出すテストコードが書かれ得る。ターゲット仮想アドレスポイントのセットに収斂するように、様々なテストケースからの仮想アドレスアクセスシーケンスが使用され得る。アプリケーションバイナリコードから機能を抽出するために他の機構が使用されてよいことを諒解されたい。

【００２６】

図4は、ネイティブバイナリ実行モデルにおける悪意のあるコード活動を検出するための方法400の実施形態を示すフローチャートである。ブロック402において、アプリケーションのためにVAFMT120が生成される。上記で説明したように、VAFMT120は、対応するハイレベルアプリケーション機能にマッピングされた複数の注目仮想アドレスを含む。ブロック404において、たとえば、ポータブルコンピューティングデバイスなど、コンピューティングデバイス上にアプリケーションがインストールされ得る。ブロック406において、HLOS106によって提供されるセキュリティサポートのために、アプリケーションが登録され得る(たとえば、登録アプリケーション112)。ブロック408において、アプリケーションが起動されてよく、応答して、CPU102は、アプリケーションバイナリコード108を実行し得る。登録アプリケーション112が実行されているとき、HLOS106は、アプリケーションの実行中のプロセスをインターセプトし得る(ブロック410)。ブロック412において、HLOS106は、対応するVAFMT120を使用して、機能注目ポイントを、実行されているときに検出および記録し得る。ブロック414において、記録されたポイントは、悪意のある攻撃を検出および解決するために、悪意のあるコード検出アルゴリズム116に提供され得る。悪意のあるコード検出アルゴリズム116は、シグネチャベースのアルゴリズム、パターンマッチングアルゴリズムを含むこと、または機械学習もしくは他の技法を用いることがある。このようにして、悪意のあるコード検出アルゴリズム116は、VAFMT120を使用して、入力として受信する仮想アドレスの意味を提供し得る。

【００２７】

VAFMT120はHLOS106の制御下にあるので、HLOS106によって実行されるアプリケーションバイナリコード108の仮想アドレスの任意の変換/ランダム化(たとえば、アドレス空間レイアウトランダム化(ASLR))が、実行中のアプリケーションの有効仮想アドレスとの同期を維持するために、VAFMT120における仮想アドレスに適用され得る。一実施形態では、VAFMT120とともにJavaScriptコードおよびアプリケーション実行から集められた情報は、ハイレベルウェブ/JavaScript機能情報を提供することができ、この情報は悪意のあるコード検出アルゴリズム116に供給され得る。任意の悪意のある挙動を検出すると(ブロック416)、HLOS106は、アプリケーション/レンダラ/JavaScriptプロセスを休止し、ユーザのダイアログボックスを開いて、潜在的危険について警告し、ユーザに進むことに関する指示を求めることができる。ユーザが依然として進むことを望んでいる場合、ブラウザプロセスがHLOS106によって再開され得る。ユーザが進むことを望んでいない場合、HLOS106はユ

ーザに、タブを閉じるか、もしくは何らかの他のウェブサイトにナビゲートするように要請することがあり、またはHLOS106は、その実行インスタンス(ブラウザタブ)のためのプロセスを終了することがある。

【００２８】

VAFMT120は、たとえば、オーバージエア(OTA)更新を介して、アプリケーションバイナリコード110バージョンが変わったときに更新され得る。これらの更新により、HLOS106は任意の登録アプリケーション112に対して更新バイナリにより準備ができているようになる。更新バイナリは、同じ注目ポイントに対する新しい仮想アドレスをもたらし得る。

【００２９】

HLOS106ならびにマッピングテーブル120および122が、たとえば、仮想機械118(図1)を伴う管理されたランタイム実行モデルをサポートするように構成されてもよいことを諒解されたい。この点について、上記で説明した統合プラットフォーム機構は、システム100が、実行中のアプリケーションバイナリコード108からの所望のまたはターゲットハイレベル機能情報を決定することを可能にする。管理されたランタイム実行モデルの例示的な実施形態が図5～図10に示されている。

【００３０】

管理されたランタイムまたは仮想機械実行を伴う実施形態では、JavaScriptソースおよび/またはJavaScriptソースのためのバイトコード/ジャストインタイム(JIT)バイナリが、別のテーブル(たとえば、識別子-アドレスマッピングテーブル(IVAMT)122)の助けをかりて、仮想機械(VM)ヒープの異なる部分から読み取られ得る。IVAMT122は、VMヒープの重要な境界の仮想メモリアドレスを含む。それは、仮想機械118またはアプリケーションバイナリ108の様々な機能ポイントの仮想アドレスが維持され得る他のタイプのエントリをさらに含み得る。IVAMT122は、一般に、アプリケーション実行中に更新され、かつ/または動的に決定され得る特定の機能ポイントの仮想アドレスに使用され得ることを諒解されたい。この点について、IVAMT122は、機能ポイントを仮想アドレスにマッピングし得る。一方、VAFMT120は、静的に定義された仮想アドレスを機能的意味にマッピングし得る。したがって、VAFMT120は、アプリケーション実行中に変化しなくてよいが、たとえば、コンピューティングデバイスに対するオーバージエア(OTA)更新によって、更新されることがある。他の種々のテーブルがVAFMT120およびIVAMT122に関連付けられ得ることをさらに諒解されたい。種々のテーブルは、仮想アドレスではないパラメータ値または設定にマッピングされた様々なマクロまたはパラメータ名を含み得る。

【００３１】

図9の実施形態では、例示的なVMヒープ構造900の様々な外部および/または内部境界に関して仮想メモリアドレス901が識別される。図9に示すように、VMヒープ構造900は、たとえば、fromフィールド912、toフィールド914、コードフィールド902、マップフィールド904、大型オブジェクトフィールド906、旧データフィールド908、および旧ポインタフィールド910を含む、様々な内部および/または外部境界を識別する複数のデータフィールドを含み得る。VMヒープは、ネイティブシステムヒープにおいて割り振られた、VM管理されたメモリ領域である。当技術分野で知られているように、VMヒープにおいて、VMは、たとえば、コード(たとえば、JavaScriptソース)、バイトコード、中間コード、JITedバイナリ、実行中に作成されたオブジェクト、ならびにすべての他の関連するハウスキーピング情報およびプログラム(たとえば、JavaScriptプログラム)の実行に使用される内部データ構造を割り振り、割振り解除する、メモリ管理の抽象化を実行する。図9にさらに示すように、VMヒープ領域は、VMが記憶する物のタイプに応じて様々な下位領域(たとえば、910、908、906、904、902、912、および914)を含み得る。下位領域912および914は、最初に作成されたオブジェクトを含めるために使用され得、任意のガベージコレクション活動は、下位領域912から914に生のオブジェクトをスワップし、その逆も同様である。一実施形態では、下位領域902は、JavaScriptソース、バイトコード、中間コード、およびJITedバイナリ/アセンブリコードを保存するために使用され得る。下位領域904は、プログラム(たとえば、JavaScriptプログラム)の実行中にVMによって作成されたオブジェクトに関連

付けられたいくつかの内部データ構造を維持するために使用され得る。下位領域906は、所定のサイズ(たとえば、1MB)よりも大きい任意の種類のアイテム(コード、オブジェクト)を維持するために使用され得る。下位領域908および910は、ガベージコレクションの複数のサイクルにわたって存続してきたオブジェクトおよびデータを維持することができ、下位領域908は、定数値を有するオブジェクトに焦点を当て、下位領域910は、他のオブジェクトを指すオブジェクトに焦点を当てる。

【００３２】

動作中、HLOS106は、VMヒープに関してメモリ割振りが変化したときにIVAMT122における仮想メモリアドレス901を識別し、動的に更新し得る。JavaScript仮想機械118が、関数がアクティブになるまでヒープにおけるソースを維持することを諒解されたい。管理されたランタイムまたは仮想機械実行モデルは、VMヒープからJavaScriptソースおよび/またはバイトコード/JITコードを識別することを伴い得る。JavaScriptソースを保持しているVMヒープオブジェクトは、新しい書込みがないかを追跡されてよく、仮想機械118によって受信された新しいJavaScriptソースが識別され得る。識別されたJavaScriptソースは、信頼できるゾーン114におけるアルゴリズム116に提供されてよく、そこでは、JavaScriptコードから様々な特徴を抽出し、任意の悪意のある挙動を検出するためにそれらを使用する。JavaScriptコードから抽出される特徴の例には、以下の特徴または他の特徴が含まれ得る。ドキュメントオブジェクトモデル(DOM)修正および感度関数、いくつかの評価、いくつかのストリング、スクリプト長、ストリング修正関数、難読化解除のための「built-ins」など。信頼できるゾーン115は、任意の悪意のある活動を決定するために、抽出された特徴を悪意のあるコード検出アルゴリズム116に供給し得る。

【００３３】

いくつかの実施形態では、JITバイナリ/バイトコードのみが利用可能であるとき、特徴は、それらから抽出され、次いで、悪意のあるコード検出アルゴリズム116に送られ得る。たとえば、HLOS106は、ハイレベルJavaScriptアーティファクトを表すバイトコード/JITコードシーケンスのライブラリを維持し得る。これらのアーティファクトとVMコード空間におけるJavaScript関数からのバイトコード/JITコードストリームの任意のマッチングが記録され、悪意のある特性の決定のために悪意のあるコード検出アルゴリズム116に渡され得る。

【００３４】

図5および図6は、管理されたランタイムまたは仮想機械実行中に使用されるIVAMT122およびVAFMT120の例示的な実施形態を示す。図5は、対応するアプリケーションバイナリコード108へのVMコード空間の割振りに関係するターゲット機能の論理マッピング500を示す。この実施形態では、アプリケーションソースコード110は、「AllocateVMCodeSpace」関数のためのコードを含む。図5に示すように、ソースコード110における第1のポイントは、バイナリコード108における仮想アドレス502にマッピングされ得る。ソースコード110における第2のポイントは、バイナリコード108における仮想アドレス504にマッピングされ得る。例示的な実装形態では、実行中のVMが実行する必要がある新しいJavaScriptソースコードを取得し、現在のVMヒープコード空間(902)に十分な空間がないと判断されたとき、関数AllocateVMCodeSpaceが呼び出され得る。この関数は、新しいJavaScriptコードのサイズを測り、VMがJavaScriptソース、関連するバイトコードもしくは中間コードおよび/またはJITedバイナリを保存できるように、VMヒープコード空間がサイズを拡大する必要がある量を決定し得る。決定されたサイズに基づいて、AllocateVMCodeSpace関数は、mmap()、malloc()、calloc()、またはrealloc()など、システムアロケータ関数を使用して、ネイティブプラットフォームのヒープにおけるVMヒープコード空間の割り振られた空間を拡大し得る。mmap()関数は、好ましくはアドレス開始時に、メモリにファイル記述子によって指定された他のオブジェクトからのオフセットで開始するバイトのシーケンスをマッピングするPOSIX準拠Unixシステムコールである。mmap()関数は、オブジェクトがマッピングされる実際の場所を戻す。malloc()、realloc()、calloc()およびfree()は、C/C++プログラミング言語での動的メモリ割振りのための手動メモリ管理を実行するためのC標

準ライブラリにおける関数のグループを含む。バイナリコード108における注目ポイントの仮想アドレス502および504は、VAFMT120における列302に直接配置され得る。仮想アドレスによって表される異なる注目ポイントの機能的意味は、VAFMT120の列304においてマクロ名として列挙され得る。検出アルゴリズム116(図1)は、VAFMT120の列304においてマクロによって表される機能を明確に理解し得る。VAFMT120における特定の行の(列304における)マクロ名は、プロセッサ(たとえば、CPU102)が(列302における)その仮想アドレスポイントにおけるアプリケーションのバイナリ命令を実行するときに、実行されている機能をはっきりと識別し得る。このようにして、注目ポイントに関する仮想アドレスの実行統計値、カウントおよびプロファイルを知ることによって、検出アルゴリズム116は、ハイレベルアプリケーションバイナリによって実行されている機能を完全に理解する。マッピングは直接的に、仮想アドレス302と、マクロ(304)によって表され、処理または検出を実行する検出アルゴリズム116によって理解される機能的意味との間であってよく、それによって、その仮想アドレス注目ポイントにおける実際のバイナリ命令を知る必要を減らし得ることを諒解されたい。

【００３５】

仮想アドレスおよびマクロの意味とともに表される注目ポイントは、オフラインで決定され、次いで、特定のアプリケーションバイナリに対してVAFMT120にポピュレートされ得る。多くのタイプのアプリケーションは、利用可能なマッチング参照ソースコードを有し得る。たとえば、マッチング参照ソースコードは、普及しているオープンソースプロジェクトから開発された一般に利用可能なアプリケーション(たとえば、ブリンク/Chromiumベースのブラウザ、Webkitベースのブラウザ、Dalvik、ART、RenderScriptなど、Androidプラットフォームにおける様々な仮想機械)に利用可能であり得る。利用可能なマッチング参照ソースコードを有するアプリケーションの場合、商用アプリケーションバイナリにおける注目ポイントの仮想アドレスを、それらの注目ポイントのソースコードにおける対応する表現/ステートメントに関して決定するために、様々なオフライン機構が使用され得る。

【００３６】

注目ポイントの仮想アドレスのオフライン決定のための例示的な実施形態について説明する。注目機能を実施するソースコード110におけるいくつかの重要かつ有用な関数が、マッチング参照ソースコードにおいて識別され得る。合わせて特定の一意の機能を表すポイントの一意のセットを形成するために、ソースコード110内の様々なポイントが手動で決定され得る。これは、機能に対して完全なソースコード110の機能全体を一意に表すソースコード110内のサンプルポイントのセットと同等であり得ることを諒解されたい。ソースコード110はコンパイルされ、アセンブルされ、実際の商用サードパーティ製アプリケーションと同等である参照アプリケーションにリンクされ得る。両方のバイナリ(参照および商用サードパーティ)は、同じソースコード110から生じ、同様のビルド技法(たとえば、コンパイル、アセンブル、リンク)およびツールチェーンを使用し得る。当分野で知られているように、オープンソースアプリケーションは、自由に利用可能なGCCまたはLLVMツールチェーンを使用し得る。コンパイラ、アセンブラ、およびリンカツールは、参照バイナリアプリケーションを生成するために使用されてよく、ソースコードにおける重要なポイントに対応する仮想アドレスポイントに留意することができる。注目ポイントの仮想アドレスは、バイナリアプリケーションがビルドされる(コンパイルされる、アセンブルされる、リンクされる)ソースコード110における注目ポイントの直接的マッピングを含み得るので、商用サードパーティバイナリにおける仮想アドレス注目ポイントを識別するために商用バイナリと比較するために、参照バイナリがオフラインで使用され得る。商用サードパーティバイナリにおける注目ポイントの仮想アドレスを決定するために、他のオフラインまたは他の技法が使用されてよいことをさらに諒解されたい。一実施形態では、図2は、ソースコード110における異なる注目ポイント(201、203、205)が、バイナリ108における対応する仮想アドレス(202、204、206)にどのように直接マッピングされ得るかを示す。

【００３７】
図6は、図5のVAFMT120と例示的なIVAMT122との間の論理マッピング600を示す。VAFMT120は、実行が注目され、追跡されているバイナリアプリケーションにおける固定および既知の注目ポイントの仮想アドレスを含む。これらの仮想アドレスは、バイナリアプリケーションが変わるときはいつでも更新され得る。IVAMT122は、バイナリアプリケーションが実行されるときに作成または更新される特定のポイントの仮想アドレスを含み、これは動的であり、動的なアイテム(たとえば、ランタイムバッファ開始または終了ポイント)の仮想アドレスを表し得る。VAFMT120の左手列(302)は、仮想アドレスを含み、右手列(304)は、その仮想アドレスポイントにおけるバイナリコード108に存在する機能的説明を示し得る。このようにして、VAFMT120は、仮想アドレスを機能的意味にマッピングする。一般に、IVAMT122は逆方向を含む。この場合、機能的意味またはマクロ名が知られており、システムは、機能的意味またはマクロ名604が実装されているか、またはバイナリアプリケーションの実行インスタンスにおいて利用可能である仮想アドレス602を決定する。IVAMT122における仮想アドレスは、ランタイムにおいて決定される動的な値を含み得る。動的に割り振られたバッファ(または仮想機械ヒープもしくはそのサブ空間)の開始および終了が決定される場合、動的バッファ/ヒープ-空間割振りをしているバイナリアプリケーションにおける関数内の注目ポイントの仮想アドレスがVAFMT120から取得され得る。これらの関数の実行は、VAFMT120における仮想アドレスの実行を検出することによって決定され得る。さらに、バッファ/ヒープ-空間割振りの開始/終了仮想アドレスは、これらの関数から呼び出されたシステムメモリ割振り関数を検出することによって決定され得る。バッファ/ヒープ-空間割振りのこれらの決定された開始/終了仮想アドレスは、IVAMT(122)において更新され得る。

【００３８】
図7は、VMヒープコード空間に対するガベージコレクションの影響、およびどのようにJavaScriptソースが、仮想機械118のガベージコレクション活動が存在する場合に一貫して決定され得るかを示す。新しいオブジェクトの割振りおよび無効の(すなわち、使用されていない)オブジェクトの割振り解除がランタイムまたは仮想機械118によって明示的に処理され得るので、ガベージコレクションは、管理されたランタイムまたは仮想機械の不可欠な活動であることを諒解されたい。管理されたVMヒープから無効の(すなわち、使用されていない)オブジェクトを回収する活動は、ガベージコレクションと呼ばれる。この点について、必要とされないScriptオブジェクトまたは他のオブジェクトが回収されるとき、VMヒープは再編成され、既存のオブジェクトはあちこち移動し、新しいオブジェクト割振りのための空間を作るために圧縮され得る。図7は、VMヒープコード空間704aに対するそのようなガベージコレクション活動の影響を示す。VMヒープコード空間704aは、JavaScriptオブジェクトJS1、JS2、JS3、およびJS4を含む。ガベージコレクション事象の後、それらは圧縮され、JavaScriptオブジェクトJS3が除去されてよく、JavaScriptオブジェクトJS3は、ガベージコレクタによって必要とされないものまたは無効のものとして検出され、したがってVMヒープコード空間704bから回収(削除)されている。ただし、VMヒープにおけるオブジェクトのいかなるそのような動き(たとえば、除去、圧縮など)によっても、JavaScriptオブジェクトが存在する場所を決定する仮想アドレス開始および終了ロケーションは変わる。例示的な方法では、各ガベージコレクション活動の後、VMヒープおよびヒープ内の様々な空間(図9)に関して図5および図6に示す仮想アドレス決定機構を再実行することによって、仮想アドレスは変更され得、それによって、ガベージコレクション中にScriptオブジェクトが移動した場合に仮想アドレスは新しい値により更新され得る。図8に示すように、カーネルは、ガベージコレクション中に発生しているオブジェクト移動およびそれらが移動する距離を追跡し得る。オブジェクトが移動したアドレスオフセットを追跡することによって、VMヒープコード空間におけるJavaScriptオブジェクトの開始および終了の仮想アドレス値が更新され得る。同様に、VMヒープの様々なコード空間のIVAMT122における仮想アドレスは、図9に示すVMヒープの様々なサブ空間の割振り/割振り解除/移動を追跡することによって更新され得る。

【００３９】
図10は、管理されたランタイムまたは仮想機械実行モデルにおける悪意のあるコード活動を検出するための方法1000の実施形態を示すフローチャートである。図10のブロック1002、1004、1006、1008、および1010において表されるステップまたは機能は、一般に、図4の方法に関連して上記で説明したブロック402、404、406、408、および410に対応し得ることを諒解されたい。ブロック1012において、方法1000は、VMヒープアロケータ/ディアロケータ関数の注目ポイント仮想アドレスを、実行されたときに検出する。ブロック1014に示すように、実行がVMヒープアロケータ/ディアロケータ関数の内部であることが検出されたとき、方法1000は、カーネルのシステムアロケータ/ディアロケータ関数へのエントリVMを検出し、システムメモリ割振り/割振り解除を記録することができる。それに基づいて、方法1000は、VMのヒープの開始/終了仮想アドレスを計算および決定し得る。VMヒープの特定の割振り領域(たとえば、コード空間、大型オブジェクト空間など)のために同様の機構を実装することによって、VMヒープ内の特定の下位領域(たとえば、コード空間、大型オブジェクト空間など)の開始/終了仮想アドレスが決定され得る。ブロック1016に示すように、JavaScriptソースコードオブジェクトを記憶するために使用されるVMヒープ空間がブロック1014において決定されると、方法1000は、VMヒープ内のJavaScriptオブジェクトの開始を決定するために(バイナリでの)スクリプトオブジェクトヘッダシグネチャ/パターンを使用し得る。JavaScriptオブジェクトの長さは、ヘッダから抽出され、JavaScriptソースコード全体を抽出するために使用され得る。ブロック1018に示すように、JavaScriptソースコードは、たとえば、悪意のある挙動を検出するために検出アルゴリズム116によって使用される特定の注目特徴を抽出するために使用され得る。ブロック1020において、JavaScriptコードの悪意のある挙動が、たとえば、ブロック1018においてJavaScriptソースから抽出された特徴に基づいて決定され得る。

【００４０】
上述のように、VAFMT120は、最初に、オフライン方式で構成され、コンピューティングシステム100(図1)に提供され得る。一実施形態では、アプリケーションバイナリコード108の新バージョンがコンピューティングシステム100にとって利用可能になったとき、VAFMT120は、同様に、オフライン方式で更新され、コンピューティングシステム100に、たとえば、通信ネットワークを介して提供され得る(「オーバージエア(OTA)更新」と呼ばれる)。このようにしてVAFMT120を更新することは、頻繁に更新されるバイナリアプリケーションにとって不利であり得る。アプリケーションバイナリコード108の更新バージョンにおけるバイナリコードの比較的大きい部分は変わらないままであり得ることを諒解されたい。VAFMT120において識別される機能注目ポイント304は、アプリケーションバイナリコード108および/またはバージョン間で変わらないことがあるバイナリコードの比較的限られた部分を含み得る。

【００４１】
たとえば、コンパイラ動作および/または設定は、まれに変わることがあり、バイナリコードにおける様々なモジュールは、モジュール間で同様のまたは所定のオフセットを維持し得る。図12～図16は、アプリケーションバイナリコード108の新バージョンまたは更新バージョンがインストールされたときにVAFMT120における仮想アドレスを自動的に更新するためにコンピューティングデバイス100において実装され得る様々な機構を示す。

【００４２】
これらの機構が様々なタイプのアプリケーションおよび/または使用事例に関するVAFMT120のOTA更新の必要を減らし得ることを諒解されたい。たとえば、セキュリティアプリケーションの文脈では、これらの機構は、同じ発生コードベースに基づくウェブブラウザアプリケーションに対する最もよくあるタイプの更新の多くに関するOTA更新の必要をなくし得る。既存のウェブブラウザアプリケーションは、週次または月次でバイナリアプリケーションコードを更新し得る。新しいバイナリバージョンの仮想アドレスは、機能注目ポイント304に関係する特定のモジュールに関してソースコードが変わっていないときでも変わり得る。この場合、仮想アドレスは、機能注目ポイント304以外のアプリケーション

の部分におけるソースコードの変更、またはアプリケーションの他の部分においてアクセスされる変数タイプおよびデータ構造タイプ(たとえば、C++クラス、C-構造、集合など)の変更がある場合に変わり得る。さらに、コンパイラ、アセンブラ、およびリンカオプションのいくつかの種類の変更は、アプリケーションの他の部分の仮想的変更をもたらし得る。

【００４３】

図12は、アプリケーションバイナリコード108の新バージョンまたは更新バージョンがインストールされたときにVAFMT120を自動的に更新するためにコンピューティングデバイス100において実装され得る例示的な機構の実施形態を示す。図12に示すように、VAFMT120は、メタデータ1200および1つまたは複数の疑似バイナリコードテンプレート1202により補足され得る。以下でより詳細に説明するように、メタデータ1200および疑似バイナリコードテンプレート1202は、HLOS106が、アプリケーションバイナリコード108が新バージョンにより更新されたときに機能注目ポイント304の新しい仮想アドレス302を決定することを可能にし得る。

【００４４】

疑似バイナリコードテンプレート1202は、局所変数のためのメモリおよび疑似レジスタにおける記憶ロケーションのための記号表現を使用する演算ステートメントのシーケンスを含むことを諒解されたい。疑似バイナリコードテンプレート1202は、それらの目的を示す様々なカテゴリーの疑似レジスタを使用し得る。一実施形態では、ArgumentReg#は、サブルーチンに引数を渡す疑似レジスタを示し得る。ReturnRegは、サブルーチン呼出しから戻るときの戻りアドレスを含み得る。ProgCounterは、プロセッサのプログラムカウンタが指す現在アドレスを含み得る。ReturnValueReg#は、呼出し側コードにサブルーチン呼出しからの値を戻すために使用されるレジスタを示し得る。演算は、変数または記憶ロケーションであり得る入力および出力を伴うプロセッサにおけるアセンブリ演算の忠実な表現(close representation)を含み得る。たとえば、AddWord変数は、サイズ4バイトまたは1ワードのオペランドの加算演算を示し得る。LoadWord変数は、所定のサイズ(たとえば、4バイトまたは1ワード)を有するメモリからの値をロードすることを示し得る。LoadByte変数は、所定のサイズ(たとえば、1バイト)を有するメモリからの値をロードすることを示し得る。branchEQは、以前の比較演算が、比較されているオペランドの等しさをもたらしている場合に、オペランドとして提供されるターゲットに分岐する条件付き分岐を含み得る。アドレス指定モードまたはアドレス計算は、ロード演算または記憶演算から分離され得る。一実施形態では、ベースレジスタおよびオフセットを伴うロード演算は、2つの演算、すなわち、疑似レジスタに一定のオフセット値を加算することによって最終アドレスを計算する加算演算、および後続の、計算された最終アドレスを含む疑似レジスタを使用する実際のロード演算に分割され得る。これは、様々な形式のアドレス指定モードが更新アプリケーションバイナリによって使用される場合に最も一般的な形式による表現を維持するために行われ得る。定数である演算引数は、有効範囲の定数を符号化するために必要とされるビットの数によって表され得る。

【００４５】

たとえば、定数「Const8bits」は、オペランドが8ビットによって符号化され得る任意の有効値であることを示す演算のためのオペランドとして使用され得、したがって、許容される値の有効な動的範囲を決定し得る。いくつかのオペランドは、ハードコーディングされた定数(たとえば、値「8」を示す「#8」)であり得る。直接分岐演算のオペランドは、現在のプログラムカウンタからのオフセット(たとえば、(「ProgCounter + #Const20bits」、または「ProgCounter + #12」))として表され得る。疑似バイナリコードテンプレート1202は、これらまたは他の演算ステートメントを使用して注目機能を実施し得る。たとえば、マッチング機能またはモジュールを介して正確な機能を実施する新しい更新バイナリにおける領域を識別するために演算ステートメントが使用され得ることを諒解されたい。マッチングモジュールは、疑似バイナリコードテンプレート1202のフォーマットおよび表現とアプリケーションの実際のバイナリの両方を理解するように構成される。マッチ

ングモジュールは、マッチングを検出するために演算のウィンドウ内で演算ごとの比較を実行するか、または制御データフローおよび制御データフロー領域内の演算を比較のために使用し得る。

【００４６】

様々なマッチング技法が使用され得る。疑似バイナリコードテンプレート1202における演算ステートメントは、静的単一代入(SSA)表現を使用することができ、SSA表現では、特定の疑似レジスタ変数が1回だけ代入され、それによって、演算ステートメント間の真の依存を明らかにする。SSA表現は、アプリケーションの更新バイナリにおける機能領域のマッチングの改善を可能にし得る。「疑似」という用語は、表現がバイナリ実行可能ではなく、プロセッサの実際のアセンブリ命令、レジスタ、およびアドレス指定モードを使用せず、バイナリコードにアセンブルされないことを指す。疑似バイナリコードテンプレート1202は、アプリケーションの更新バイナリにおける注目機能を検出するためにマッチングモジュールがテンプレートパターンおよび指針として使用する機能参照を提供する。疑似バイナリコードテンプレート1202の実際のフォーマットおよび表現は実装依存であり、様々な他の代替形態が使用されてよいことを諒解されたい。他の実施形態では、いくつかの実装形態は、実際のアセンブリ命令表現またはバイナリアプリケーションが実行されるCPU102のためのアセンブリ表現に似ている表現を使用し得る。

【００４７】

上記で説明したように、HLOS106は、登録アプリケーションのリスト112を維持し得る。登録アプリケーションごとに、HLOS106は、機能注目ポイント304の仮想アドレス302を含むテーブル(たとえば、VAFMT120、IVAMT122)を維持する。図12に示すように、VAFMT120における1つまたは複数の仮想アドレス302が、疑似バイナリコードテンプレート1202に関連付けられ得る。図12の実施形態では、疑似バイナリコードテンプレート1202は、一意の機能(documentWrite関数)を表す機能注目ポイント304の特定のセットの仮想アドレス302のセットに関連付けられる。疑似バイナリコードテンプレート1202は、documentWrite関数をカバーするバイナリコードと一般的に同等の疑似コード命令を含む。一実施形態では、疑似バイナリコードテンプレート1202は、プロセッサ命令セットアーキテクチャ(ISA)を使用しなくてよく、実際のバイナリコードにアセンブルされる必要がない。疑似バイナリコードテンプレート1202は、アセンブリ演算と同様の演算ステートメントを使用し、疑似レジスタおよび記号参照をストレージのために使用し得る。そのような演算ステートメントのシーケンスの使用によって、疑似バイナリコードテンプレート1202は、アプリケーションの実際のバイナリにおいて実施される注目機能(たとえば、documentWrite関数)と同じまたは同等である、それが表す注目機能(たとえば、上記の例での「documentWrite」関数の機能)を実施し得る。コンピューティングシステム100は、任意の数の疑似バイナリコードテンプレート1202を含み得ることを諒解されたい。異なる疑似バイナリコードテンプレート1202の数は、機能注目ポイントの異なるセットを通じて、VAFMT120において捕捉された異なる機能すべてが、新しいアプリケーションバイナリコードがインストールされたときにそれがカバーする関数ポイントの仮想アドレスを更新するために使用される少なくとも1つの代表的な疑似バイナリコードテンプレート1202を有するような数であり得る。

【００４８】

一実施形態では、疑似バイナリコードテンプレート1202は、一般的な形式のターゲットアセンブリ命令、1つまたは複数の疑似レジスタ、およびメモリにおける特定の参照ポイントを表す一般的なベース(たとえば、グローバルヒープまたはスタック、シンボル/変数名)からのメモリアクセスオフセットを含み得る。メタデータ1200は一般に、たとえば、バイトオフセットを使用する仮想アドレス自由表現(virtual-address free representation)を含む。仮想アドレス(0x3473fac8)のメタデータ1200は、バイトオフセット(BASE2 = BASE0 + 74709704)を含む。仮想アドレス(0x3473fad4)のメタデータ1200は、バイトオフセット(BASE2 + 12)を含む。仮想アドレス(0x3473fae8)のメタデータ1200は、バイトオフセット(BASE2 + 32)を含む。このメタデータは、「document_write」機能を一意に表す3つの仮想アドレス注目ポイントのセットに対応する一意のセットを形成し得ることを諒解

されたい。

【００４９】

疑似バイナリコードテンプレート1202は、最初にオフライン方式で生成され、コンピューティングシステム100に提供され、デバイスのセキュアなストレージに記憶され得る。疑似バイナリコードテンプレート1202は、たとえば、機能注目ポイント304によってカバーされる領域におけるコードおよび/またはデータ構造の顕著な変更があるときのみ更新される必要があり得ることを諒解されたい。これらのタイプの変更は、比較的まれ(たとえば、6ヵ月に1回)であり得る。このタイプおよび他のタイプの更新は、OTA更新を介して実施され得る。これは、たとえば、週次/月次から、6ヵ月に1回だけ疑似バイナリコードテンプレート1202のOTA更新を行うことのみへの、仮想アドレスのOTA更新の大幅な減少を可能にする。

【００５０】

既存の登録アプリケーションの新しいバイナリバージョンの更新または再インストールが検出され得る。応答して、メタデータ1200および疑似バイナリコードテンプレート1202が、VAFMT120を自動的に更新するために使用され得る。図12に示すように、疑似バイナリコードテンプレート1202は、疑似バイナリコードテンプレート1202によって表される機能注目ポイント304(ひいては、この特定の疑似バイナリコードテンプレートが表す仮想アドレス注目ポイント)が位置する新しいアプリケーションにおけるバイナリコードの領域1206をパターンマッチングするために使用され得る。メタデータ1200は、アプリケーションバイナリコード108の更新バージョン1204において探索される領域1206に焦点を当てるために使用され得る。一意の機能の機能注目ポイント304の元のベース(BASE0)からの相対的OFFSETを使用することによって、焦点領域(Focused Region)1206(たとえば、ベース、BASE2の前および後の所定のパーセンテージ)上で探索する最初の試みが行われ得る。多くのタイプの頻繁な更新では、これらの相対的オフセットが依然として接近していることを諒解されたい。図12にさらに示すように、マッチングが検出されたとき、新しい仮想アドレスが新しいバイナリから取得され得、VAFMT120は、新しい仮想アドレスを反映するために更新され得る。1つまたは複数の機能注目ポイント304が新しいバイナリにおけるマッチングをもたらさない場合、コンピューティングシステム100は、OTA更新を開始してよく、または他の実施形態では、特定の注目機能および関連する仮想アドレスを、特定の機能の重要性に基づいてVAFMT120から削除してよい。

【００５１】

図13は、(グレーアウトされたボックスによって表される)更新仮想アドレスとともに図12からのVAFMT120を示す。DOCUMENT_WRITE_FUNCTION_START注目ポイント304に対応する仮想アドレス302は、新しい仮想アドレス(0x3133b61c)に更新されている。DOCUMENT_WRITE_1注目ポイント304に対応する仮想アドレス302は、新しい仮想アドレス(0x3133b62c)に更新されている。DOCUMENT_WRITE_2注目ポイント304に対応する仮想アドレス302は、新しい仮想アドレス(0x3133b640)に更新されている。図13にさらに示すように、仮想アドレスに対応するメタデータ1200も更新され得る。図13に示すように、新しい仮想アドレス(0x3133b61c)のメタデータ1200は、「BASE2 = BASE0 + 74709000」に更新されている。これは、アプリケーションの更新バイナリにおける2つの注目機能の間(すなわち、「KERNEL_ALLOCATOR_FUNCTION」と「DOCUMENT_WRITE_FUNCTION」との間)にわずかな相対位置変更があったことを示す。変更は比較的わずかであり得る。たとえば、変更は、それらの間の74709704バイトの元の総距離のうちの704バイトの縮小であり得る。したがって、探索により、2つの注目機能の間のベースオフセットメタデータ(すなわち、74709704バイト)の前および後のいくつかのトレランスとともに焦点を当てられて、探索領域を狭くすることによって効果的なマッチングが可能になる。新しい仮想アドレス(0x3133b62c)のメタデータ1200は、BASE2 + 16に更新されている。新しい仮想アドレス(0x3133b640)のメタデータ1200は、BASE2 + 36に更新されている。

【００５２】

図14および図15は、DOCUMENT_WRITE関数に関係する機能注目ポイント304のセットに関

連付けられた疑似バイナリコードテンプレート1202の例示的な実施形態を示す。機能注目ポイント304のセットは、DOCUMENT_WRITE_FUNCTION_STARTモジュール、DOCUMENT_WRITE_1モジュール、およびDOCUMENT_WRITE_2モジュールを含む。図14に示すように、セットにおける機能注目ポイント304の各々は、疑似バイナリコードテンプレート1202内の「疑似バイナリ命令注目ポイント」を形成する特定の疑似コード命令に直接関連付けられる。疑似バイナリコードテンプレート1202内のこれらの「疑似バイナリ命令注目ポイント」は、「疑似バイナリ注目ポイント」と直接マッチングした更新アプリケーションバイナリにおける特定のバイナリ命令に応じた、アプリケーションバイナリの更新バージョンにおける新しい仮想アドレス注目ポイントと現在のVAFMT120における仮想アドレス注目ポイントの1対1のマッピングを含む。図14に示すように、DOCUMENT_WRITE_FUNCTION_STARTモジュールは、最初の2つの呼出し側保存済み疑似レジスタ(CallSave0、CallSave1)および戻りレジスタ(ReturnReg)を保存する「プッシュ」演算に関連付けられる。その後に、後続のLoadWord演算によって必要とされるアドレスを計算するAddWord演算がある。AddWord演算は、プログラムカウンタで8ビットに適合すべき定数値を加算し、疑似レジスタreg0に結果を保存する。後続のLoadWord演算は、値のロード元のアドレスとしてreg0のアドレスを直接使用する。アプリケーションのための実際のバイナリでは、8ビットの定数を有するAddWordは、アドレス指定モードの一部としてLoadWord命令に直接含まれ得る。「Const8bits」は、8ビットに適合する任意の定数値を有するオプションを可能にする。ロードされた値は、疑似レジスタreg1において維持され、疑似レジスタreg2に値をロードする第2のLoadWord演算のためのアドレスとして使用される。DOCUMENT_WRITE_FUNCTION_STARTによって示される機能注目ポイントの場合、「プッシュ」演算は、この疑似バイナリコードテンプレート1202における「疑似バイナリ命令注目ポイント」である。

【００５３】

DOCUMENT_WRITE_1モジュールは、疑似レジスタ(reg0)において維持され、疑似レジスタreg1に保存される値の16ビットによる論理シフト左演算に関連付けられる。それは次いで、定数値「4」で加算され、疑似レジスタreg2に保存され、次いで疑似レジスタreg2は、疑似レジスタ(reg3)に値がロードされる際のロード元のアドレスとして使用される。実際のバイナリロード命令の場合、アドレス指定モードは、定数値4による加算を直接実行することができ、したがって、AddWordおよびLoadWordは、単一のロード命令によって表され得ることに留意されたい。reg3における値は、プログラムカウンタ値(PC)にさらに加算されて、呼び出されたルーチンに第1の引数として渡すために使用される第1の引数レジスタ「ArgumentReg0」にバイト値がロードされる際のロード元のアドレスである疑似レジスタreg4における最終アドレスが作成される。その後、20ビットに適合することができる値であるオフセットにあるアドレスへの直接的分岐がある。しかしながら、直接的分岐命令の前に、直接的分岐がアプリケーションの異なる部分への制御ができるようになった後に(ReturnRegを適切に設定することによって)戻るためのアドレスを保存するAddWord命令がある。「論理シフト左」演算は、DOCUMENT_WRITE_1によって示される機能注目ポイントに関するこの疑似バイナリコードテンプレート1202における「疑似バイナリ命令注目ポイント」である。

【００５４】

DOCUMENT_WRITE_2モジュールは、プログラムカウンタで8ビットに適合することができる定数値を加算し、疑似レジスタreg0において結果を維持するAddWord演算に関連付けられる。疑似レジスタreg0は次いで、疑似レジスタ(reg2)に値がロードされる際のロード元のアドレスとして使用される。その後に、疑似レジスタ(reg2)およびプログラムカウンタの現在値を加算し、疑似レジスタreg1において結果を維持する別のAddWord演算がある。疑似レジスタreg1は次いで、直接的分岐命令を通じて後続のサブルーチン呼出しに値を渡すために使用されるArgumentReg0に値がロードされる際のロード元のアドレスとして使用される。実際のバイナリロード命令の場合、アドレス指定モードは、定数値による加算を直接実行することができ、したがって、AddWordおよびLoadWordは、アプリケーションの実際のバイナリでの単一のロード命令によって表され得ることに留意されたい。LoadWord

演算の後、20ビットに適合することができる値であるオフセットにあるアドレスへの直接的分岐がある。しかしながら、直接的分岐命令の前に、直接的分岐がアプリケーションの異なる部分への制御ができるようになった後に(ReturnRegを適切に設定することによって)戻るためのアドレスを保存するAddWord命令がある。サブルーチンの呼出しの後、比較の2つのセットおよび疑似バイナリコードテンプレート1202内の近くのロケーションへの分岐がある。両方の比較を、第1のサブルーチン戻り値レジスタ(ReturnValueReg0)に対して行って、サブルーチンによって戻された特定の値(「0」および「1」)をチェックし、戻された値に基づいて、局所的にそれぞれBranchEQ演算およびBranchNE演算を使用して分岐を行う。分岐ターゲットアドレスは、現在のプログラムカウンタ値からの定数オフセットとして提供される。プログラムカウンタでConst8bitsオペランドを加算するAddWord演算は、DOCUMENT_WRITE_2によって示される機能注目ポイントに関するこの疑似バイナリコードテンプレート1202における「疑似バイナリ命令注目ポイント」である。アプリケーションの実際のバイナリは、(「ldr r1、[pc,#80]として)単一の実際のバイナリ命令への疑似バイナリコードテンプレートマッチングにおいて、LoadWord演算とともにこのアドレス計算演算(AddWord)を有することができ、この場合、全体的または部分的のいずれかにおいて「疑似バイナリ命令注目ポイント」がマッチングする実際のバイナリ命令が、アプリケーションのバイナリの新バージョンにおける更新仮想アドレスを決定する命令になることに留意されたい。

【００５５】

図15は、アプリケーションバイナリコード108の更新バージョン1204のマッチング領域1206における同等の対応するバイナリコードへの疑似バイナリコードテンプレート1202における疑似コード命令の各々のマッチングを示す。動作中、疑似バイナリコードテンプレート1202が領域1206とマッチングしたとき、機能注目ポイント304とマッチングするバイナリコードにおける対応する命令の仮想アドレスが、新しい仮想アドレスになり、VAFMT120において更新される。新しいベースおよびオフセットが新しい仮想アドレスに基づいて計算され得、メタデータ1200が更新され得る。

【００５６】

図16は、アプリケーションバイナリコード108の新バージョンまたは更新バージョンがインストールされたときにVAFMT120を自動的に更新するためにコンピューティングデバイス100において実装される方法1600の実施形態を示す。ブロック1602において、上記で説明したように、コンピューティングシステム100に、HLOS106に登録されたアプリケーションのための仮想アドレスマッピングテーブル120が記憶され得る。VAFMT120は、HLOS106中のセキュアなメモリに記憶され得る。図12に示すように、VAFMT120は、登録アプリケーションのアプリケーションバイナリコード108における対応するターゲットアプリケーション機能(機能注目ポイント304)にマッピングされた仮想アドレス302の複数のセットを含み得る。アプリケーションバイナリコード108の更新バージョン1204を受信したこと(決定ブロック1604)に応答して、仮想アドレスマッピングテーブル120における仮想アドレス302の複数のセットのうちの1つまたは複数に関連付けられた対応する疑似バイナリコードテンプレート1202が決定され得る(ブロック1606)。上述のように、一実施形態では、疑似バイナリコードテンプレート1202は、最初に、当初のVAFMT120とともにシステム100へのオーバージエア(OTA)更新を通じて、またはシステム100上にコード/データをダウンロードおよびインストールする任意の他の手段によって獲得され得る。これらの疑似バイナリコードテンプレート1202とVAFMT120の両方が、システム100中の、HLOS106およびカーネルによってアクセス可能なロケーションに記憶され得る。実際の記憶ロケーションは実装依存である。様々なレベルのセキュリティ保護またはセキュアなメモリ構成が、記憶ロケーションに対して考慮され得、実装選択に依存する。疑似バイナリコードテンプレート1202は、たとえば、既存のテンプレートのうちの1つまたは複数がアプリケーションの更新バイナリにおいてマッチングを発見することができないときに更新され得る。注目領域におけるアプリケーションコードの大規模な変更、または上記で説明した他の種類の変更に起因して、ミスマッチが発生し得る。そのような状況の間、更新された疑似バイナリコードテ

ンプレート1202および更新されたVAFMT120が、システム100にOTAダウンロードおよびインストールされ得る。決定ブロック1608において、疑似バイナリコードテンプレート1202は、アプリケーションバイナリコード108の更新バージョン1204を探索し、同等のバイナリ命令に疑似コード命令をマッチングするために使用される。マッチングが発見されたとき、ブロック1610において、バイナリ命令に対応する新しい仮想アドレスが決定される。ブロック1612において、仮想アドレスマッピングテーブル120は、新しい仮想アドレスおよび対応する更新ベース/オフセットメタデータ1200により更新され得る。

【００５７】

図16に示すように、ブロック1606、1608、1610、および1612は、異なる疑似バイナリコードテンプレート1202のすべてに対して、疑似バイナリコードテンプレート1202のすべてがマッチングされ、VAFMT120における仮想アドレスのすべてが更新されるまで繰り返され得る。決定ブロック1611において、方法1600は、疑似バイナリコードテンプレート1202のすべてが処理されたかどうかを判断し得る。「yes」の場合、方法1600はブロック1613において終了することができる。「no」の場合、ブロック1606において新しい疑似バイナリコードテンプレート1202が選択され得る。決定ブロック1608において、マッチングするバイナリシーケンスが特定の疑似バイナリコードテンプレート1202のアプリケーションの更新バイナリにおいて識別されると、方法1600は、マッチングのために次の疑似バイナリコードテンプレート1202に対して繰り返すことができる。ある繰り返しにおいて、アプリケーションの更新バイナリにおける疑似バイナリコードテンプレート1202に関するマッチングがない場合、疑似バイナリコードテンプレート1202によって表される注目機能がVAFMT120から削除され得るかどうかが最初に判断される(決定ブロック1607)。それが削除され得る(機能の重要性が低いことを含む、様々な理由に起因し得る)場合、この注目機能に関する仮想アドレス注目ポイントのエントリすべてがVAFMT120から削除され得(ブロック1605)、次の疑似バイナリコードテンプレート1202に関するマッチングを探索するために、ブロック1606で繰り返しが続く。しかしながら、機能(ひいては、疑似バイナリコードテンプレート1202)が重要であり、削除されるべきではない場合(ブロック1609)、自動的更新機構は失敗し、その場合、仮想アドレスおよび/または疑似バイナリコードテンプレート1202の完全なオーバージエア(OTA)更新が実行され得る。これは、(たとえば、より低い頻度で、6ヵ月に1回発生する)アプリケーションの更新バイナリの大幅な変更/修正がある場合を表し得る。

【００５８】

図17～図23は、オフセットベースの仮想アドレスマッピング方式を使用してターゲットアプリケーション機能を検出するためのシステムおよび方法の様々な実施形態を示す。一般に、オフセットベースの仮想アドレスマッピング方式は、仮想アドレスオフセットを使用する、対応するハイレベルターゲットアプリケーション機能へのアプリケーションバイナリコード108における仮想アドレスのマッピングを伴う。オフセットベースの仮想アドレスマッピングは、同じアプリケーションの複数のプロセスの同時実行の場合のターゲットアプリケーション機能の検出を可能にするために特に有用であり得ることを諒解されたい。一実施形態では、ターゲットアプリケーション機能は、ウェブブラウザアプリケーションの複数のブラウザタブまたはインスタンスの同時実行の際に検出され得る。さらに、オフセットベースの仮想アドレスマッピング方式は、再配置可能なアドレスを有する動的共有ライブラリを用い得る。

【００５９】

図17は、オフセットベースの仮想アドレスマッピング方式1700の例示的な実施形態のアーキテクチャおよび/または動作を示す。図17に示すように、オフセットベースの仮想アドレスマッピング方式1700は、2つの異なるタイプの仮想アドレス-関数マッピングテーブル、すなわち、アプリケーション固有VAFMT1702および1つまたは複数のプロセス固有VAFMT1714によってサポートされる2段階方式を伴う。各登録アプリケーション112は、対応するアプリケーションバイナリコード108のために生成され得るアプリケーション固有VAFMT1702を有し得る。アプリケーション固有VAFMT1702は、対応するターゲットアプリケーシ

ョン機能にマッピングされる、アプリケーションバイナリコード108における複数の仮想アドレスオフセットを含む。この点について、アプリケーション固有VAFMT1702は、上記で説明したように、実際の仮想アドレスを直接定義するのではなく、仮想アドレスオフセットを含む。仮想アドレスオフセットは、仮想アドレス範囲内のロケーション差異を定義することを諒解されたい。一実施形態では、仮想アドレスオフセットは、アプリケーションバイナリコード108の開始から定義されたベース仮想アドレスに対するロケーション差異、または他の実施形態では、ターゲットアプリケーション機能間の仮想アドレス範囲内の相対的差異を定義し得る。

【００６０】

図17にさらに示すように、アプリケーションがロードされるとき、0/Sアプリケーションローンチャおよびローダ1704が、同時に実行される、アプリケーションの2つ以上のインスタンスまたはアプリケーションに関連付けられたプロセスの2つ以上のインスタンス(まとめて「アプリケーションプロセスインスタンス」1708と呼ばれる)を起動し得る。たとえば、アプリケーションがウェブブラウザを含む場合、アプリケーションプロセスインスタンス1708は、ウェブブラウザの複数のインスタンスまたは複数のブラウザタブを含み得る。図17の実施形態では、0/Sアプリケーションローンチャおよびローダ1704は、3つのアプリケーションプロセスインスタンス1708a、1708b、および1708cを起動している。アプリケーションプロセスインスタンス1708ごとに、対応するプロセス固有VAFMT1714が生成される。アプリケーションプロセスインスタンス1708aのために、プロセス固有VAFMT1714aが生成される。アプリケーションプロセスインスタンス1708bのために、プロセス固有VAFMT1714bが生成される。アプリケーションプロセスインスタンス1708cのために、プロセス固有VAFMT1714cが生成される。

【００６１】

図17の実施形態に示すように、カーネルモジュール1706が、アプリケーションプロセスインスタンス1708の起動に応答して、対応するプロセス固有VAFMT1714を作成し得る。アプリケーション固有VAFMT1702に記憶された仮想アドレスオフセット(参照番号1712)および0/Sアプリケーションローンチャおよびローダ1704によって提供されたベース仮想アドレス(参照番号1710)を使用して、プロセス固有VAFMT1714が生成される。たとえば、アプリケーションプロセスインスタンス1708aが起動されたとき、0/Sアプリケーションローンチャおよびローダ1704は、アプリケーションプロセスインスタンス1708aがロードされた仮想アドレスベースを提供し得る。カーネルモジュール1706は、仮想アドレスベースに仮想アドレスオフセットを加算することによって、ターゲットアプリケーション機能の実際の仮想アドレスを決定し得る。アプリケーションプロセスインスタンス1708aの計算された実際の仮想アドレスは、プロセス固有VAFMT1714aに記憶される。アプリケーションプロセスインスタンス1708bが起動されたとき、0/Sアプリケーションローンチャおよびローダ1704は、アプリケーションプロセスインスタンス1708bがロードされた仮想アドレスベースを提供し得る。カーネルモジュール1706は、仮想アドレスベースに仮想アドレスオフセットを加算することによって、ターゲットアプリケーション機能の実際の仮想アドレスを決定し得る。アプリケーションプロセスインスタンス1708bの計算された実際の仮想アドレスは、プロセス固有VAFMT1714bに記憶される。アプリケーションプロセスインスタンス1708cが起動されたとき、0/Sアプリケーションローンチャおよびローダ1704は、アプリケーションプロセスインスタンス1708cがロードされた仮想アドレスベースを提供し得る。カーネルモジュール1706は、仮想アドレスベースに仮想アドレスオフセットを加算することによって、ターゲットアプリケーション機能の実際の仮想アドレスを決定し得る。アプリケーションプロセスインスタンス1708cの計算された実際の仮想アドレスは、プロセス固有VAFMT1714cに記憶される。

【００６２】

このようにして、プロセス固有VAFMT1714a、1714b、および1714cは、アプリケーションバイナリコード108におけるターゲットアプリケーション機能にマッピングされた、それぞれアプリケーションプロセスインスタンス1708a、1708b、および1708cの実際の仮想ア

ドレスを含む。アプリケーションプロセスインスタンス1708a、1708b、および1708cの同時実行中、それぞれプロセス固有VAFMT1714a、1714b、および1714cは、上記で説明した方法でターゲットアプリケーション機能を検出するために使用される。図1～図16に関連して上記で説明した他のマッピングテーブル(たとえば、IVAMT122、「JavaScriptソースコードリストテーブル」など)の構造は変わらないままであり得ることを諒解されたい。これらのマッピングテーブルは、実際のプロセス固有仮想アドレスにより初期化され得る、アプリケーションプロセスインスタンス1708の一意のインスタンスを含み得る。一実施形態では、マッピングテーブルは、アプリケーションプロセス実行中に動的に初期化され得る。(たとえば、ASLRなどのような活動のための)仮想アドレスに対する任意のプロセス固有の調整が、プロセス固有VAFMTインスタンスに対して、かつプロセスランタイム中に動的に初期化される他のテーブル(たとえば、IVAMT122など)のために、それらがカーネル制御下にあるので、行われ得ることをさらに諒解されたい。アプリケーション固有VAFMT1702は、ASLRおよび他の活動のために調整される必要がないことがある仮想アドレスオフセットを有する。

【００６３】

図18は、アプリケーション固有VAFMT1702の例示的な実施形態を示す。列1800がVAFMT120の場合のような実際の仮想アドレスの代わりに仮想アドレスオフセットを定義することを除いて、アプリケーション固有VAFMT1702はVAFMT120と同様に構成され得ることを諒解されたい。この点について、図18は、アプリケーションバイナリコード108におけるコードに関連付けられた仮想アドレスオフセット(列1800)の、それらの仮想アドレスオフセットにおけるコードが表すそれぞれの機能的意味(列1802において識別される機能注目ポイント)への論理マッピングを示す。アプリケーション固有VAFMT1702は同様に、アプリケーションバイナリコード108が新バージョンにより更新されたときにHLOS106が機能注目ポイント(列1802)の新しい仮想アドレスオフセットを決定することを可能にするために疑似バイナリコードテンプレート1202と組み合わせて使用されるメタデータ(列1804)を含み得る。

【００６４】

図19はプロセス固有VAFMT1714の例示的な実施形態を示す。プロセス固有VAFMT1714は、アプリケーション固有VAFMT1702と同様に、それぞれ機能注目ポイントおよびメタデータを識別する列1802および1804を含み得る。図19に示すように、プロセス固有VAFMT1714は、アプリケーション固有VAFMT1702において識別される仮想アドレスオフセットの代わりに対応するアプリケーションプロセスインスタンス1708の実際の仮想アドレスを記憶するための列1900を含み得る。実際の仮想アドレスに関して列1900に記憶される値は、カーネルモジュール1706によって、アプリケーション固有VAFMT1702に記憶された仮想アドレスオフセットおよび0/Sアプリケーションローンチャおよびローダ1704によって提供されたベース仮想アドレスを使用して決定され得る。図19の例では、アプリケーションプロセスインスタンス1708は、値0x30000000を有するベース仮想アドレスからロードされ得る。カーネルモジュール1706は、このベース仮想アドレス値を受信し、応答して、アプリケーションプロセスインスタンス1708の実際の仮想アドレスを計算し得る。図18のアプリケーション固有VAFMT1702における第1の行を参照すると、EVAL_FUNCTION(列1802)は、0x373ea94(列1800)の仮想アドレスオフセット値を有し得る。アプリケーションプロセスインスタンス1708におけるEVAL_FUNCTIONの実際の仮想アドレスを計算するために、カーネルモジュール1706は、ベース仮想アドレス値(0x30000000)および仮想アドレスオフセット値(0x373ea94)を加算し得る。図19の第1の行に示すように、EVAL_FUNCTIONの計算された実際の仮想アドレスは、値0x3373ea94(値0x30000000および0x373ea94の和)を有する。図19の第2の行では、DOCUMENT_WRITE_FUNCTION_STARTの計算された実際の仮想アドレスは、値0x3473fac8(値0x30000000および0x473fac8の和)を有する。図19における残りの行の実際の仮想アドレスは、同様に式1に従って計算され、列1900に記憶され得ることを諒解されたい。

実際のVA = ベースVA + 仮想アドレスオフセット

式1

【００６５】
上記の例示的な実施形態は、実際の仮想アドレスを計算するために加算演算を用い得る。ただし、他の実施形態では、実際の仮想アドレスは、たとえば、計算するために使用される約束事、特定のオペレーションシステム/プラットフォームに関する(たとえば、上位または下位アドレスへの)メモリ割振りの指示などに応じて、ベース仮想アドレスから仮想アドレスオフセットを減算することによって取得され得ることを諒解されたい。
【００６６】
図20は、仮想アドレスオフセットを使用するURLバッファ仮想アドレスマッピングを含むアプリケーション固有VAFMT2000の別の実施形態を示す。アプリケーション固有VAFMT2000は一般に、テーブルが実際の仮想アドレスの代わりに仮想アドレスオフセットを記憶することを除いて、図11に示すVAFMT120に対応する。この点について、アプリケーション固有バージョンは、特定のデータ構造タイプ(たとえば、クラス、構造、集合)のオブジェクトを含む動的に割り振られたバッファの開始および終了の仮想アドレスを決定するために使用され得る特定のバッファアロケータ関数の仮想アドレスオフセットを有するカスタム仮想アドレスオフセットテーブルを含む。図20のURLバッファ仮想アドレスマッピングは、組込みHTTPSスタックを使用するアプリケーション(列2002)に対する個別の行を含む。第1の行は、第1のそのようなアプリケーション(アプリケーション-1)のためのURLバッファ仮想アドレスマッピングを定義し、第2の行は、第2のそのようなアプリケーション(アプリケーション-2)のためのURLバッファ仮想アドレスマッピングを定義する。列2004は、たとえば、URLデータ構造割振りを実行する関数のアプリケーション-1およびアプリケーション-2の仮想アドレスオフセット値を記憶する。列2006、2008、2010、2012、2014、および2016は、図11の列1104、1106、1108、1110、1112、および1114と直接対応する。この点について、バッファにおいて割り振られたオブジェクトのメンバー/フィールドの値は、注目ポイントである特定のフィールド/メンバーのためのテーブルにおいて維持されることもある、オフセットおよび長さフィールドを使用して決定され得ることを諒解されたい。たとえば、システムメモリアロケータ関数の実行をアロケータ関数の仮想アドレスによってカバーされる領域から追跡することによって、割り振られたバッファのサイズおよびアドレスを検出するために、バッファ割振り関数の仮想アドレスが使用され得る。バッファ開始および終了仮想アドレスが知られると、特定のデータ構造タイプに関するオブジェクトの特定のメンバー/フィールドの値を決定するために、オフセットおよび長さフィールドが使用され得る。
【００６７】
図21および図22は、それぞれアプリケーション-1およびアプリケーション-2のために生成されたプロセス固有VAFMT2100および2200に実施形態を示す。プロセス固有VAFMT2100および2200は、図20における仮想アドレスオフセット値(列2004)ならびにアプリケーション-1およびアプリケーション-2がそれぞれO/Sアプリケーションローンチャ/ローダ1704によってロードされるベース仮想アドレスを使用して生成される。上記の式1に従って計算されたアプリケーション-1の実際の仮想アドレスは、列2102(図21)に記憶され、アプリケーション-2の実際の仮想アドレスは、列2202に記憶される。
【００６８】
図23は、図17～図22に関連して上記で説明したオフセットベースの仮想アドレスマッピングを使用してターゲットアプリケーション機能を検出するための方法2300の実施形態を示すフローチャートである。ブロック2302において、アプリケーション固有VAFMT1702がアプリケーションのために生成される。上記で説明したように、アプリケーション固有VAFMT1702は、対応するハイレベルアプリケーション機能にマッピングされた複数の注目仮想アドレスオフセットを含む。ブロック2304において、たとえば、ポータブルコンピューティングデバイスなど、コンピューティングデバイス上にアプリケーションがインストールされ得る。ブロック2306において、HLOS106によって提供されるセキュリティサポートのために、アプリケーションが登録され得る(たとえば、登録アプリケーション112)。ブロック2308において、アプリケーションが起動され得る。アプリケーションを起動したこ

とに応答して、アプリケーションのインスタンスまたはアプリケーションプロセスインスタンス1708のためにプロセス固有VAFMT1714が生成され得る。プロセス固有VAFMT1714は、アプリケーション固有VAFMT1702に記憶された仮想アドレスオフセットおよびアプリケーションまたはインスタンスがロードされる際のロード元のベース仮想アドレスを使用して、ハイレベルアプリケーション機能の実際の仮想アドレスを定義する。アプリケーションバイナリコード108は、実行し始めることができる。

【００６９】

ブロック2310において、HLOS106は、アプリケーションの実行中のプロセスをインターセプトし得る。ブロック2312において、HLOS106は、プロセス固有VAFMT1714を使用して、機能注目ポイントを、実行されているときに検出および記録し得る。ブロック2314において、記録されたポイントは、悪意のある攻撃を検出および解決するために、悪意のあるコード検出アルゴリズム116に提供され得る。悪意のあるコード検出アルゴリズム116は、シグネチャベースのアルゴリズム、パターンマッチングアルゴリズムを含むこと、または機械学習もしくは他の技法を用いることがある。参照番号2318によって示されるように、ブロック2308、2310、2312、2314、および2316は、複数のアプリケーションプロセスインスタンスに対して、同時実行中のアプリケーションプロセスインスタンスに対して悪意のあるコードが検出され得るように、繰り返され得る。

【００７０】

本明細書で説明した方法ステップのうちの1つまたは複数は、上記で説明したモジュールなどのコンピュータプログラム命令としてメモリに記憶される場合があることを諒解されたい。これらの命令は、本明細書で説明した方法を実行するために、対応するモジュールと組み合わせてまたは協働して、任意の適切なプロセッサによって実行される場合がある。

【００７１】

本明細書で説明したプロセスまたはプロセスフローにおけるいくつかのステップは、当然、説明したように本発明が機能するために他のステップに先行する。しかしながら、そのような順序またはシーケンスが本発明の機能を変えない場合には、本発明は、説明したステップの順序に限定されない。すなわち、本発明の範囲および趣旨から逸脱することなく、いくつかのステップは、他のステップの前に実行され、後に実行され、または他のステップと並行して(実質的に同時に)実行される場合があることを認識されたい。場合によっては、本発明から逸脱することなく、いくつかのステップが省略される場合または実行されない場合がある。さらに、「その後(thereafter)」、「次いで(then)」、「次に(next)」などの語は、ステップの順序を限定することを意図していない。これらの語は単に、例示的な方法の説明を通して読者を導くために使用される。

【００７２】

加えて、プログラミングにおける当業者は、たとえば、本明細書におけるフローチャートおよび関連する説明に基づいて、難なく、開示した発明を実装するコンピュータコードを書くことができるか、または実装するのに適したハードウェアおよび/もしくは回路を識別することができる。

【００７３】

したがって、プログラムコード命令または詳細なハードウェアデバイスの特定のセットの開示が、本発明をどのように製作し使用すべきかについて適切に理解するために必要であるとは見なされない。特許請求されるコンピュータ実装プロセスの発明性のある機能は、上記の説明において、かつ様々なプロセスフローを示す場合がある図とともに、より詳細に説明される。

【００７４】

1つまたは複数の例示的な態様では、説明した機能は、ハードウェア、ソフトウェア、ファームウェア、またはこれらの任意の組合せにおいて実装される場合がある。ソフトウェアにおいて実装される場合、機能は、コンピュータ可読媒体上の1つまたは複数の命令またはコードとして記憶または送信されてもよい。コンピュータ可読媒体は、コンピュー

タ記憶媒体と、ある場所から別の場所へのコンピュータプログラムの転送を容易にする任意の媒体を含む通信媒体の両方を含む。記憶媒体は、コンピュータによってアクセスされ得る任意の利用可能な媒体であってもよい。例として、限定はしないが、そのようなコンピュータ可読媒体は、RAM、ROM、EEPROM、NANDフラッシュ、NORフラッシュ、M-RAM、P-RAM、R-RAM、CD-ROMもしくは他の光ディスクストレージ、磁気ディスクストレージもしくは他の磁気記憶デバイス、または、命令もしくはデータ構造の形態における所望のプログラムコードを搬送もしくは記憶するために使用され得、コンピュータによってアクセスされ得る任意の他の媒体を含み得る。

【００７５】

また、あらゆる接続が、コンピュータ可読媒体と適切に呼ばれる。たとえば、ソフトウェアが、同軸ケーブル、光ファイバケーブル、ツイストペア、デジタル加入者線(「DSL」)、または赤外線、無線、およびマイクロ波などのワイヤレス技術を使用してウェブサイト、サーバ、または他のリモートソースから送信される場合、同軸ケーブル、光ファイバケーブル、ツイストペア、DSL、または赤外線、無線、およびマイクロ波などのワイヤレス技術は、媒体の定義に含まれる。

【００７６】

本明細書で使用されるディスク(disk)およびディスク(disc)は、コンパクトディスク(disc)(「CD」)、レーザーディスク(登録商標)(disc)、光ディスク(disc)、デジタル多用途ディスク(disc)(「DVD」)、フロッピーディスク(disk)、およびブルーレイディスク(disc)を含み、ディスク(disk)は、通常、データを磁気的に再生し、ディスク(disc)は、レーザーを用いてデータを光学的に再生する。上記の組合せもまた、コンピュータ可読媒体の範囲内に含まれるべきである。

【００７７】

本発明の趣旨および範囲から逸脱することなく、本発明が関係する代替的な実施形態が、当業者には明らかになるであろう。したがって、選択された態様が図示され詳細に説明されてきたが、以下の特許請求の範囲によって定義されるように、本発明の趣旨および範囲から逸脱することなく、各態様において様々な置換および改変が行われてよいことが理解されよう。

【符号の説明】

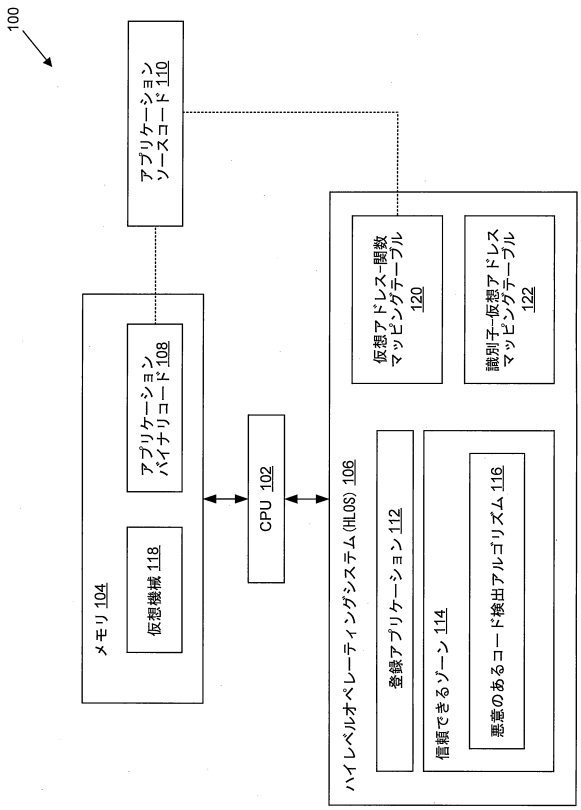
【００７８】

100　システム、コンピューティングシステム
102　中央処理装置(CPU)
104　メモリ
106　ハイレベルオペレーティングシステム(HLOS)
108　アプリケーションバイナリコード、バイナリコード、アプリケーションバイナリ、バイナリ
110　参照アプリケーションソースコード、アプリケーションソースコード、ソースコード
112　登録アプリケーションのリスト、リスト、登録アプリケーション
114　信頼できるゾーン
115　信頼できるゾーン
116　悪意のあるコード検出アルゴリズム、アルゴリズム、検出アルゴリズム
118　仮想機械、JavaScript仮想機械
120　仮想アドレス-関数マッピングテーブル、マッピングテーブル、VAFMT、仮想アドレスマッピングテーブル
122　識別子-仮想アドレスマッピングテーブル、マッピングテーブル、識別子-アドレスマッピングテーブル(IVAMT)
200　論理マッピング
201　ポイント、注目ポイント
202　仮想アドレス、バイナリコードおける仮想アドレス

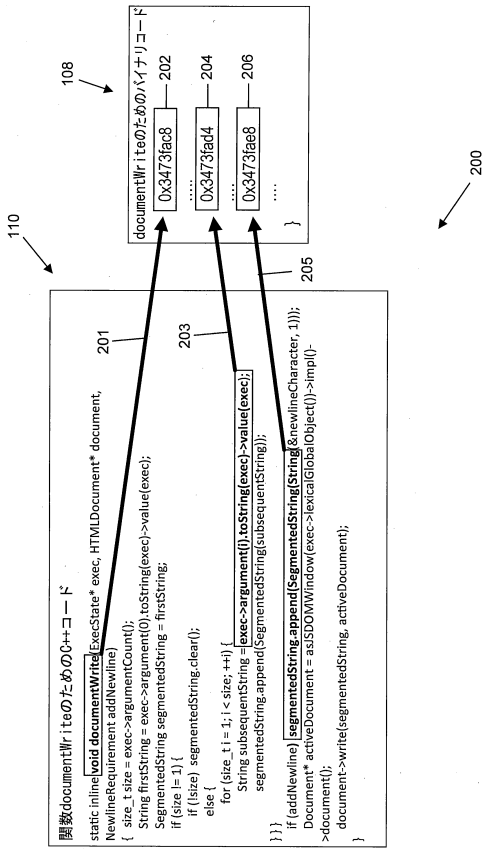
203 ポイント、注目ポイント
204 仮想アドレス、バイナリコードおける仮想アドレス
205 ポイント、注目ポイント
206 仮想アドレス、バイナリコードおける仮想アドレス
300 論理マッピング
302 列、仮想アドレス、左手列
304 列、機能注目ポイント、右手列、機能的意味を表すマクロ
400 方法
500 論理マッピング
502 仮想アドレス
504 仮想アドレス
600 論理マッピング
602 仮想アドレス
602 機能的意味またはマクロ名
702 スクリプトオブジェクトヘッダ
704a VMヒープコード空間、VMコード空間
704b VMヒープコード空間、VMコード空間
900 VMヒープ構造
901 仮想メモリアドレス
902 コードフィールド、下位領域、コード
904 マップフィールド、下位領域、マップ
906 大型オブジェクトフィールド、下位領域、大型オブジェクト
908 旧データフィールド、下位領域、旧データ
910 旧ポインタフィールド、下位領域、旧ポインタ
912 下位領域、fromフィールド
914 下位領域、toフィールド
1000 方法
1104 列
1106 列
1108 列
1110 列
1112 列
1114 列
1200 メタデータ、更新ベース/オフセットメタデータ、バイトオフセットを使用する仮想アドレス自由表現
1202 疑似バイナリコードテンプレート
1204 更新バージョン、アプリケーションバイナリの更新バージョン
1206 領域、焦点領域、マッチング領域
1600 方法
1700 オフセットベースの仮想アドレスマッピング方式
1702 アプリケーション固有VAFMT、アプリケーション固有仮想アドレス-関数マッピングテーブル(VAFMT)
1704 O/Sアプリケーションローンチャおよびローダ、O/Sアプリケーションローンチャ/ローダ
1706 カーネルモジュール
1708 アプリケーションプロセスインスタンス
1708a アプリケーションプロセスインスタンス
1708b アプリケーションプロセスインスタンス
1708c アプリケーションプロセスインスタンス
1712 VAオフセット

1714　プロセス固有VAFMT
1714a　プロセス固有VAFMT
1714b　プロセス固有VAFMT
1714c　プロセス固有VAFMT
1800　列、VAオフセット
1802　列、機能POI
1804　列、メタデータ
1900　列、実際のVA
2000　アプリケーション固有VAFMT
2002　列
2004　列
2006　列
2008　列
2010　列
2012　列
2014　列
2016　列
2100　プロセス固有VAFMT
2102　列
2200　プロセス固有VAFMT
2202　列
2300　方法
JS1　JavaScriptオブジェクト
JS2　JavaScriptオブジェクト
JS3　JavaScriptオブジェクト
JS4　JavaScriptオブジェクト

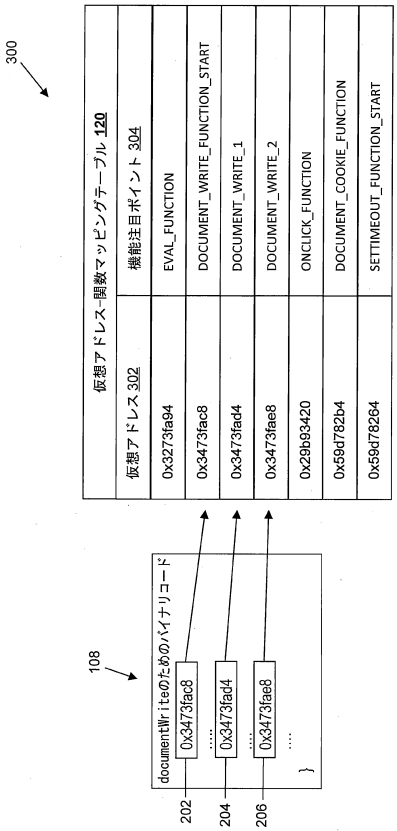
【図１】


100
アプリケーション
ソースコード 110
アプリケーション
バイナリコード 108
メモリ 104
仮想機械 118
CPU 102
ハイレベルオペレーティングシステム (HLOS) 106
仮想アドレス-関数
マッピングテーブル
120
識別子-仮想アドレス
マッピングテーブル
122
登録アプリケーション 112
信頼できるゾーン 114
悪意のあるコード検出アルゴリズム 116

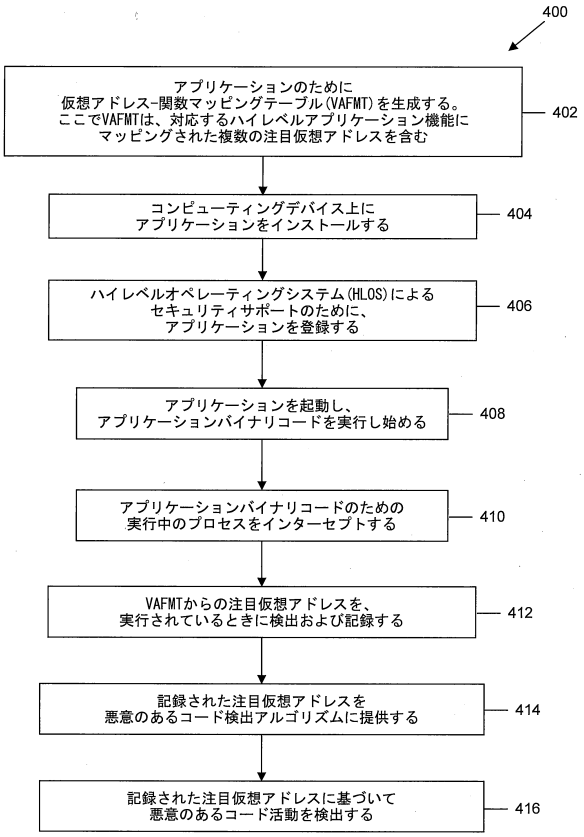
【図２】


200
108
documentWriteのためのバイナリコード
0x3473fac8
0x3473fad4
0x3473fae8
202
204
206
110
205
201
203
関数documentWriteのためのC++コード
static inline void documentWrite(ExecState* exec, HTMLDocument* document,
NewlineRequirement addNewline)
{ size_t size = exec->argumentCount();
String firstString = exec->argument(0).toString(exec)->value(exec);
SegmentedString segmentedString = firstString;
if (size != 1) {
if (!size) segmentedString.clear();
else {
for (size_t i = 1; i < size; ++i) {
String subsequentString = exec->argument(i).toString(exec)->value(exec);
segmentedString.append(SegmentedString(subsequentString));
}}}
if (addNewline) segmentedString.append(SegmentedString(String(&newlineCharacter, 1)));
Document* activeDocument = asJSDOMWindow(exec->lexicalGlobalObject())->impl()->
>document();
document->write(segmentedString, activeDocument);
}

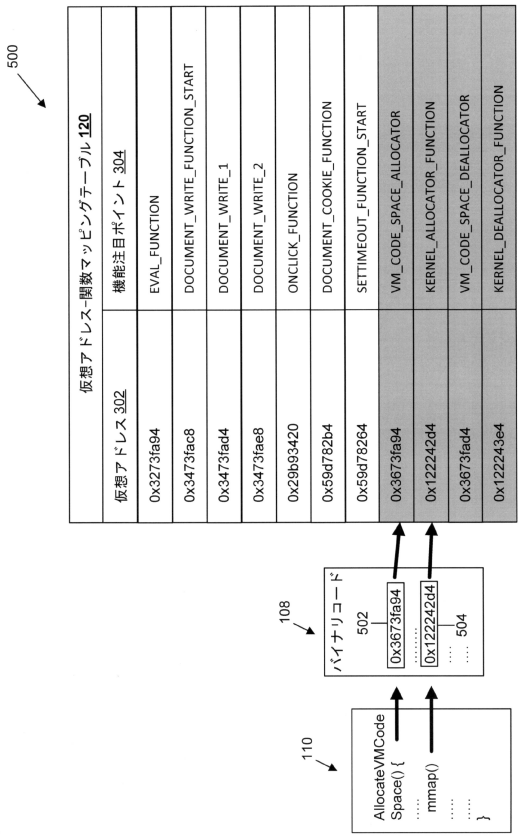
【図３】


300
仮想アドレス-関数マッピングテーブル 120
仮想アドレス 302
機能注目ポイント 304
0x3273fa94 EVAL_FUNCTION
0x3473fac8 DOCUMENT_WRITE_FUNCTION_START
0x3473fad4 DOCUMENT_WRITE_1
0x3473fae8 DOCUMENT_WRITE_2
0x29b93420 ONCLICK_FUNCTION
0x59d782b4 DOCUMENT_COOKIE_FUNCTION
0x59d78264 SETTIMEOUT_FUNCTION_START
108
documentWriteのためのバイナリコード
0x3473fac8
0x3473fad4
0x3473fae8
202
204
206

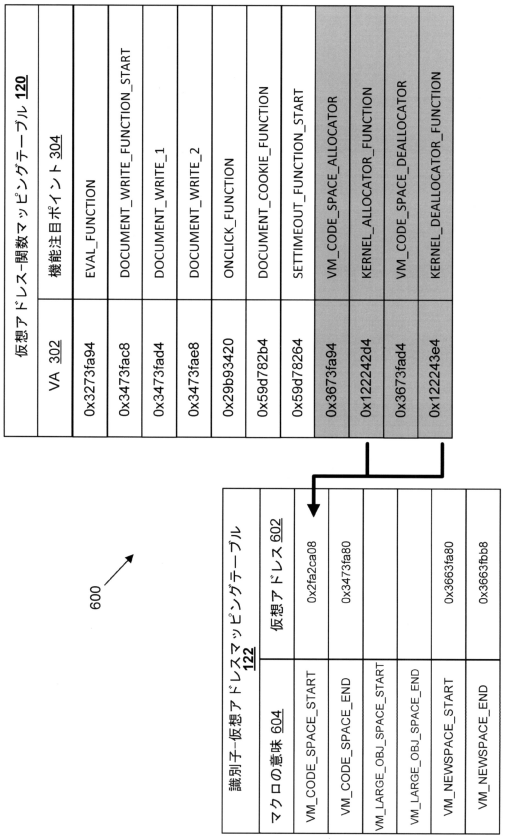
【図４】


400
アプリケーションのために
仮想アドレス-関数マッピングテーブル (VAFMT) を生成する。
ここでVAFMTは、対応するハイレベルアプリケーション機能に
マッピングされた複数の注目仮想アドレスを含む
402
コンピューティングデバイス上に
アプリケーションをインストールする
404
ハイレベルオペレーティングシステム (HLOS) による
セキュリティサポートのために、
アプリケーションを登録する
406
アプリケーションを起動し、
アプリケーションバイナリコードを実行し始める
408
アプリケーションバイナリコードのための
実行中のプロセスをインターセプトする
410
VAFMTからの注目仮想アドレスを、
実行されているときに検出および記録する
412
記録された注目仮想アドレスを
悪意のあるコード検出アルゴリズムに提供する
414
記録された注目仮想アドレスに基づいて
悪意のあるコード活動を検出する
416

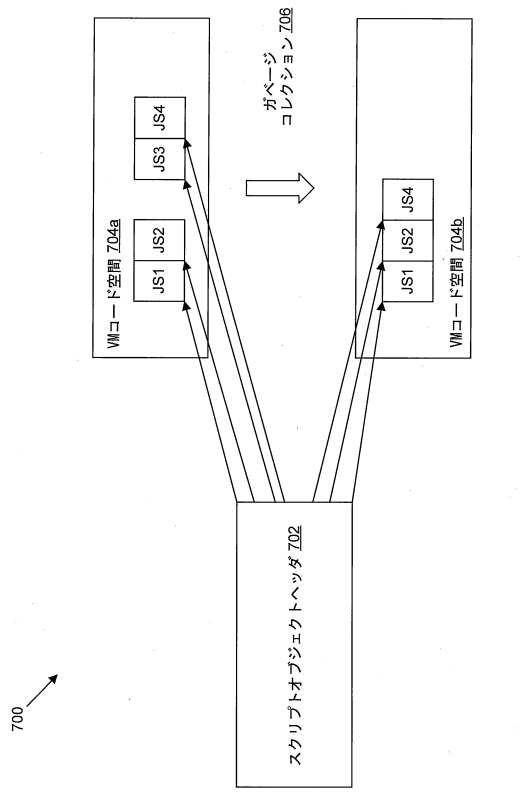
【図５】


500
仮想アドレス−関数マッピングテーブル 120
仮想アドレス 302
機能注目ポイント 304
0x3273fa94 EVAL_FUNCTION
0x3473fac8 DOCUMENT_WRITE_FUNCTION_START
0x3473fad4 DOCUMENT_WRITE_1
0x3473fae8 DOCUMENT_WRITE_2
0x29b93420 ONCLICK_FUNCTION
0x59d782b4 DOCUMENT_COOKIE_FUNCTION
0x59d78264 SETTIMEOUT_FUNCTION_START
0x3673fa94 VM_CODE_SPACE_ALLOCATOR
0x122242d4 KERNEL_ALLOCATOR_FUNCTION
0x3673fad4 VM_CODE_SPACE_DEALLOCATOR
0x122243e4 KERNEL_DEALLOCATOR_FUNCTION
108
バイナリコード
502
0x3673fa94
0x122242d4
504
110
AllocateVMCode
Space() {
mmap()
}

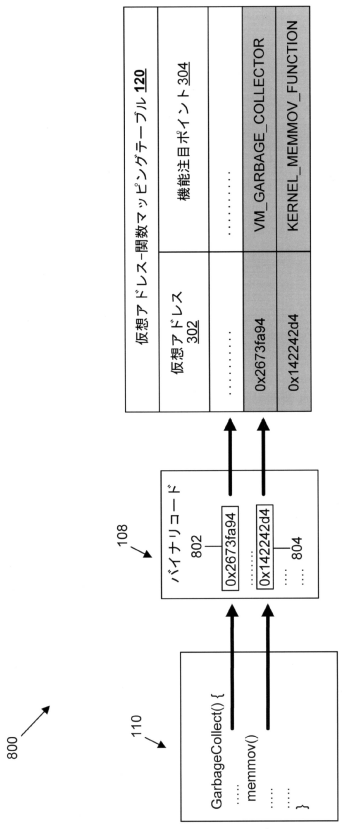
【図６】


仮想アドレス−関数マッピングテーブル 120
VA 302
機能注目ポイント 304
0x3273fa94 EVAL_FUNCTION
0x3473fac8 DOCUMENT_WRITE_FUNCTION_START
0x3473fad4 DOCUMENT_WRITE_1
0x3473fae8 DOCUMENT_WRITE_2
0x29b93420 ONCLICK_FUNCTION
0x59d782b4 DOCUMENT_COOKIE_FUNCTION
0x59d78264 SETTIMEOUT_FUNCTION_START
0x3673fa94 VM_CODE_SPACE_ALLOCATOR
0x122242d4 KERNEL_ALLOCATOR_FUNCTION
0x3673fad4 VM_CODE_SPACE_DEALLOCATOR
0x122243e4 KERNEL_DEALLOCATOR_FUNCTION
600
識別子−仮想アドレスマッピングテーブル 122
マクロの意味 604
仮想アドレス 602
VM_CODE_SPACE_START 0x2fa2ca08
VM_CODE_SPACE_END 0x3473fa80
VM_LARGE_OBJ_SPACE_START
VM_LARGE_OBJ_SPACE_END
VM_NEWSPACE_START 0x3663fa80
VM_NEWSPACE_END 0x3663fbb8

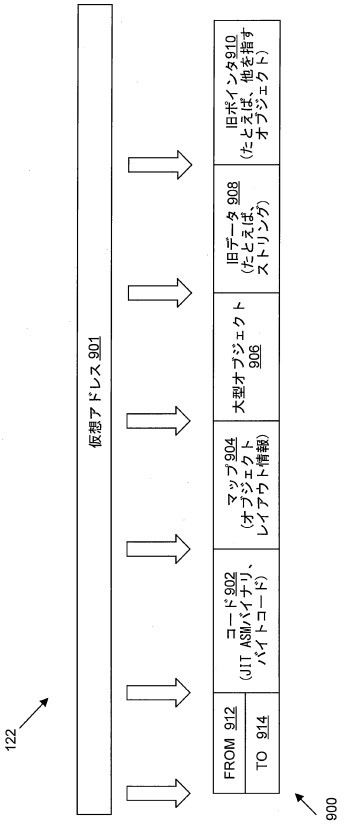
【図７】


700
VMコード空間 704a
JS1
JS2
JS3
JS4
ガベージ
コレクション 706
VMコード空間 704b
JS1
JS2
JS4
スクリプトオブジェクトヘッダ 702

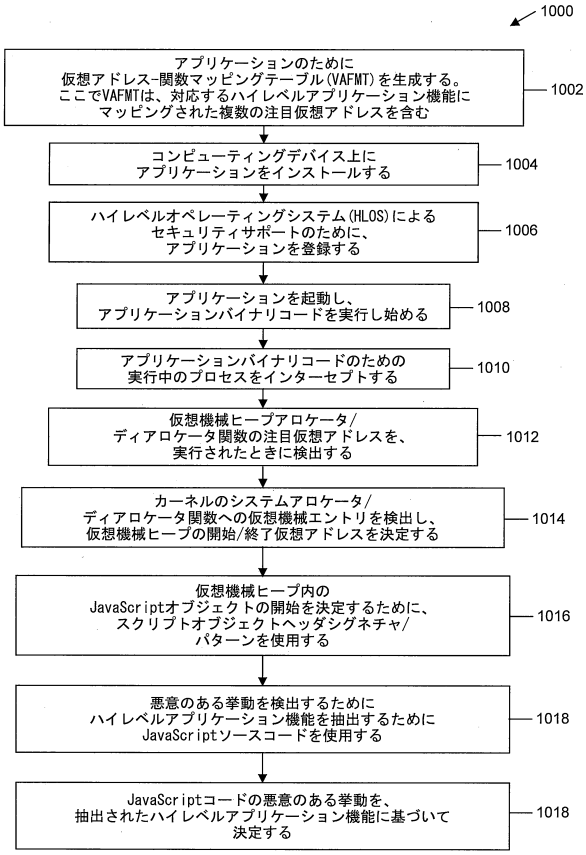
【図８】


800
仮想アドレス−関数マッピングテーブル 120
仮想アドレス 302
機能注目ポイント 304
0x26673fa94 VM_GARBAGE_COLLECTOR
0x142242d4 KERNEL_MEMMOV_FUNCTION
108
バイナリコード
802
0x26673fa94
0x142242d4
804
110
GarbageCollect() {
memmov()
}

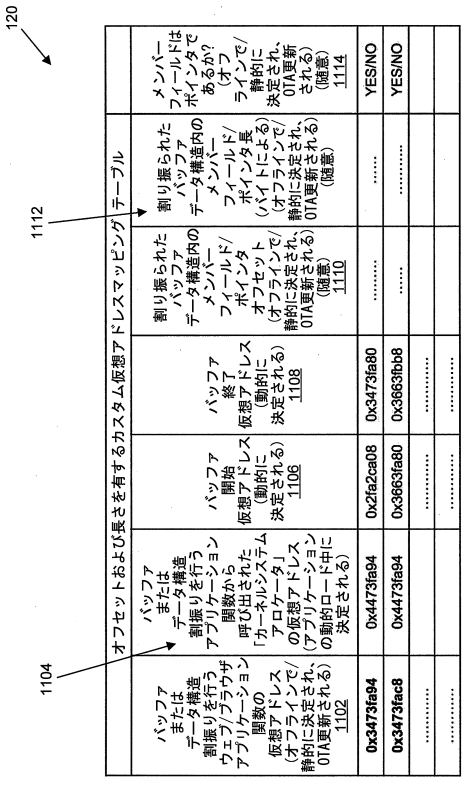
【図９】



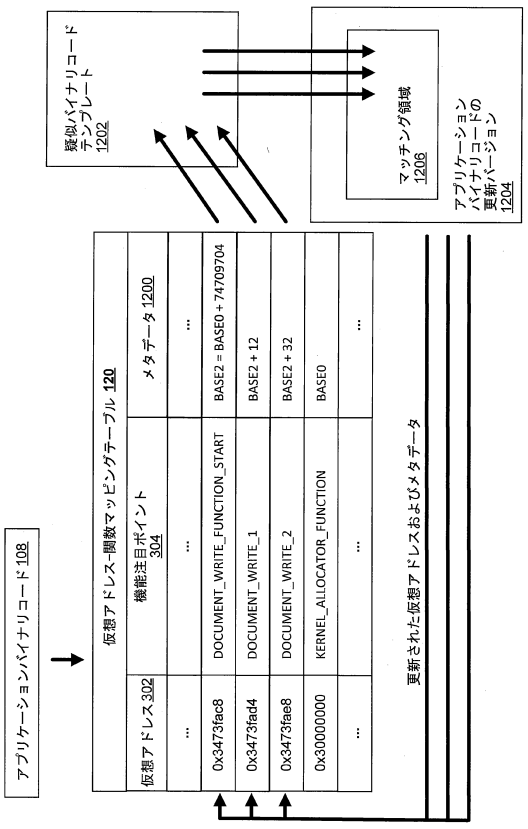
【図１０】



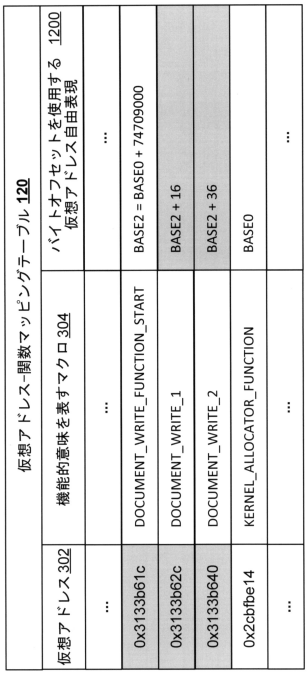
【図１１】



【図１２】



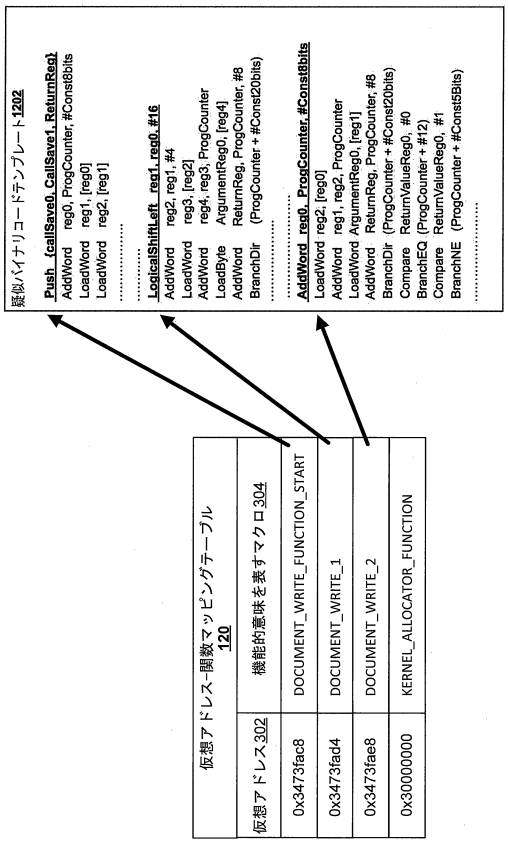
【図１３】



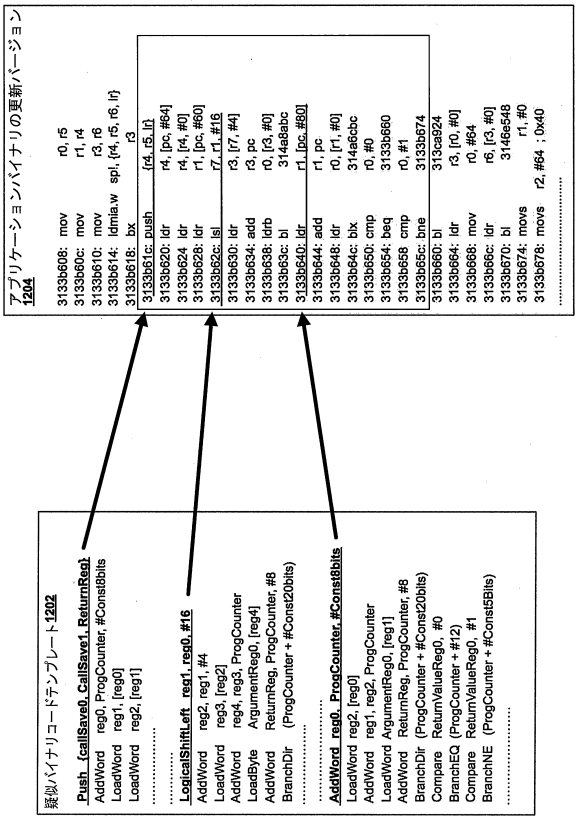
仮想アドレス-関数マッピングテーブル 120

仮想アドレス 302	機能的意味を表すマクロ 304	バイトオフセットを使用する仮想アドレス自由実現 1200
...	...	...
0x3133b61c	DOCUMENT_WRITE_FUNCTION_START	BASE2 = BASE0 + 74709000
0x3133b62c	DOCUMENT_WRITE_1	BASE2 + 16
0x3133b640	DOCUMENT_WRITE_2	BASE2 + 36
0x2cbfbe14	KERNEL_ALLOCATOR_FUNCTION	BASE0
...	...	...

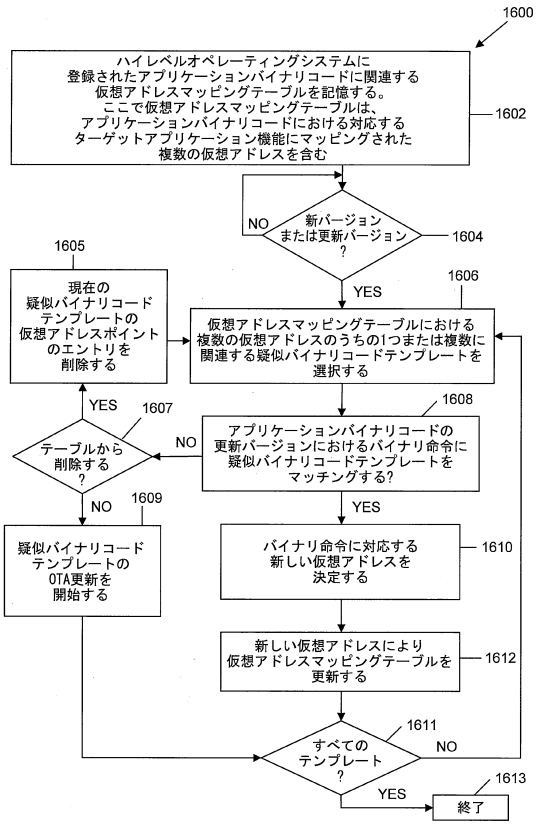
【図１４】



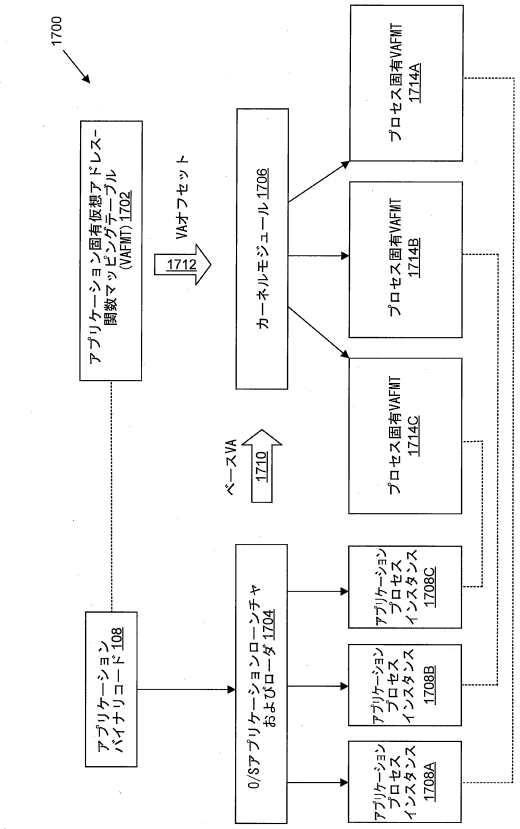
【図１５】



【図１６】



【図１７】



【図１８】

アプリケーション固有VAFMT 1702		
VAオフセット 1800	機能POI 1802	メタデータ 1804
0x373ea94	EVAL_FUNCTION	BASE1 = BASE0 + 57928340
0x473fac8	DOCUMENT_WRITE_FUNCTION_START	BASE2 = BASE0 + 74709704
0x473fad4	DOCUMENT_WRITE_1	BASE2 + 12
0x473fae8	DOCUMENT_WRITE_2	BASE2 + 32
0x1b93420	ONCLICK_FUNCTION	BASE3 = BASE0 + 28914720
0x1b934d4	ONCLICK_1	BASE3 + 180
0x9d78264	SETTIMEOUT_FUNCTION_START	BASE4 = BASE0 + 165118564
0x673fa94	VM_CODE_SPACE_ALLOCATOR	BASE5 = BASE0 + 108264084
0x673faD0	CALL_KERNEL_ALLOCATOR	BASE5 + 60
0x0000000	KERNEL_ALLOCATOR_FUNCTION	BASE0
0x673fbd4	VM_CODE_SPACE_DEALLOCATOR	BASE6 = BASE0 + 108264404
0x673fbf8	CALL_KERNEL_DEALLOCATOR	BASE6 + 36
0x22243e4	KERNEL_DEALLOCATOR_FUNCTION	BASE7 = BASE0 + 35800036

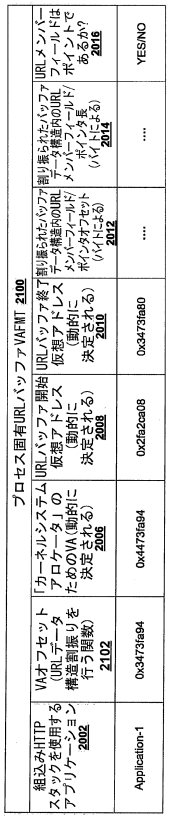
【図１９】

プロセス固有VAFMT 1714	
実際のVA 1900	機能POI 1802	メタデータ 1804
0x3373ea94	EVAL_FUNCTION	BASE1 = BASE0 + 57928340
0x373fac8	DOCUMENT_WRITE_FUNCTION_START	BASE2 = BASE0 + 74709704
0x373fad4	DOCUMENT_WRITE_1	BASE2 + 12
0x373fae8	DOCUMENT_WRITE_2	BASE2 + 32
0x31b93420	ONCLICK_FUNCTION	BASE3 = BASE0 + 28914720
0x31b934d4	ONCLICK_1	BASE3 + 180
0x39d78264	SETTIMEOUT_FUNCTION_START	BASE4 = BASE0 + 165118564
0x373fa94	VM_CODE_SPACE_ALLOCATOR	BASE5 = BASE0 + 108264084
0x373faD0	CALL_KERNEL_ALLOCATOR	BASE5 + 60
0x30000000	KERNEL_ALLOCATOR_FUNCTION	BASE0
0x373fbd4	VM_CODE_SPACE_DEALLOCATOR	BASE6 = BASE0 + 108264404
0x373fbf8	CALL_KERNEL_DEALLOCATOR	BASE6 + 36
0x322243e4	KERNEL_DEALLOCATOR_FUNCTION	BASE7 = BASE0 + 35800036

【図２０】

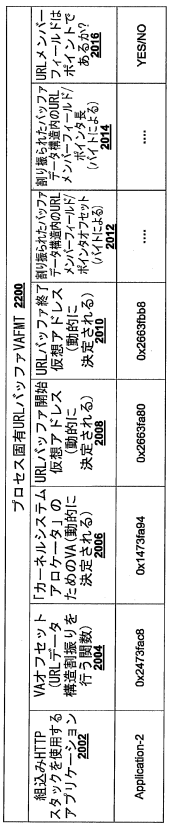
アプリケーション固有URLバッファVAFMT 2000								
組込みHTTPスタックを使用するアプリケーション 2002	VAオフセット（URLデータ構造割振りを行う関数） 2004	「カーネルシステムアロケータ」のためのVA（動的に決定される） 2006	URLバッファ開始仮想アドレス（動的に決定される） 2008	URLバッファ終了仮想アドレス（動的に決定される） 2010	割り振られたバッファ内のURLデータ構造へのオフセット（バイトによる） 2012	割り振られたバッファ内のURLデータ構造へのURLメンバーフィールド/メンバーフィールドのポインタ長（バイトによる） 2014	URLメンバーフィールドはポイントであるか? 2016
アプリケーション-1	0x473fa94						YES/NO
アプリケーション-2	0x473fac8						YES/NO
....							

【図２１】



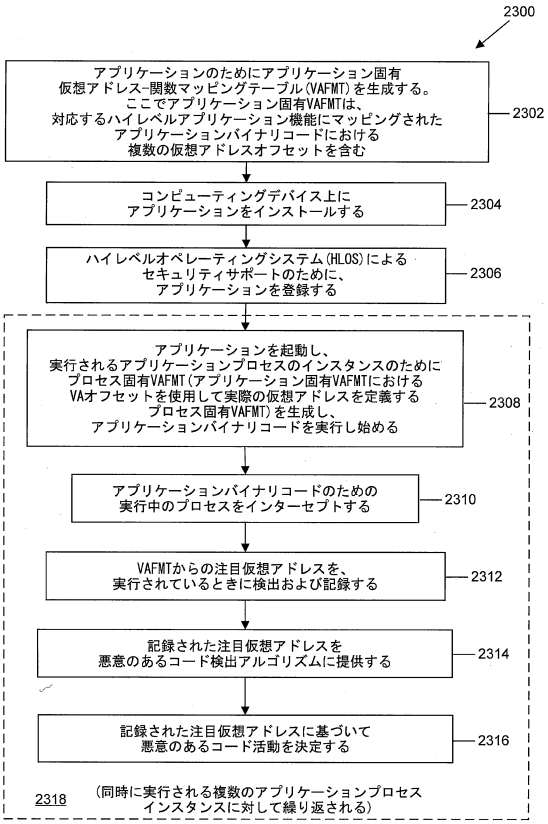
プロセス固有URLバッファVAFMT 2100						
組込みHTTPスタックを使用するアプリケーション 2002	VAオフセット（URLデータ構造割振りを行う関数） 2102	「カーネルシステムアロケータ」のためのVA（動的に決定される） 2006	URLバッファ開始仮想アドレス（動的に決定される） 2008	URLバッファ終了仮想アドレス（動的に決定される） 2010	割り振られたバッファデータ構造内のURLメンバーフィールドへのポインタオフセット（バイトによる） 2012	割り振られたバッファデータ構造内のURLメンバーフィールドへのポインタ長（バイトによる） 2014	URLメンバーフィールドはポイントであるか？ 2016
Application-1	0x34731e94	0x4473fa94	0x2fa2ea08	0x3473fa80	....	....	YES/NO

【図２２】



プロセス固有URLバッファVAFMT 2200						
組込みHTTPスタックを使用するアプリケーション 2002	VAオフセット（URLデータ構造割振りを行う関数） 2004	「カーネルシステムアロケータ」のためのVA（動的に決定される） 2006	URLバッファ開始仮想アドレス（動的に決定される） 2008	URLバッファ終了仮想アドレス（動的に決定される） 2010	割り振られたバッファデータ構造内のURLメンバーフィールドへのポインタオフセット（バイトによる） 2012	割り振られたバッファデータ構造内のURLメンバーフィールドへのポインタ長（バイトによる） 2014	URLメンバーフィールドはポイントであるか？ 2016
Application-2	0x2473fac8	0x1473fa94	0x26663fa80	0x26663fbb8	....	....	YES/NO

【図２３】



フロントページの続き

早期審査対象出願

(72)発明者　サジョ・サンダー・ジョージ
アメリカ合衆国・カリフォルニア・９２１２１・サン・ディエゴ・モアハウス・ドライヴ・５７７５

審査官　稲垣　良一

(56)参考文献　米国特許出願公開第２００２／００３２８０４（ＵＳ，Ａ１）
米国特許第６６８１３３１（ＵＳ，Ｂ１）
特表２０１５－５２５９３１（ＪＰ，Ａ）
米国特許出願公開第２０１１／００８２９６２（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　２１／５０　－　２１／５７
Ｇ０６Ｆ　１１／３６
Ｇ０６Ｆ　９／４６　－　９／５４
Ｇ０６Ｆ　１２／００　－　１２／０６
ＩＥＥＥ Ｘｐｌｏｒｅ
ＴＨＥ ＡＣＭ ＤＩＧＩＴＡＬ ＬＩＢＲＡＲＹ