(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 3 785 157 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
29.05.2024 Bulletin 2024/22

(21) Application number: **19719705.6**

(22) Date of filing: **06.04.2019**

(51) International Patent Classification (IPC):
G06F 21/55 *(2013.01)*

(52) Cooperative Patent Classification (CPC):
G06F 21/554; G06F 2221/2135

(86) International application number:
PCT/US2019/026213

(87) International publication number:
WO 2019/209500 (31.10.2019 Gazette 2019/44)

(54) MITIGATING TIMING ATTACKS VIA DYNAMICALLY SCALED TIME DILATION

VERRINGERUNG VON TIMING-ANGRIFFEN DURCH DYNAMISCH SKALIERTE ZEITDILATATION

ATTÉNUATION D'ATTAQUES DE SYNCHRONISATION PAR DILATATION TEMPORELLE À ÉCHELLE DYNAMIQUE

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **24.04.2018 US 201815961849**

(43) Date of publication of application:
03.03.2021 Bulletin 2021/09

(73) Proprietor: **Microsoft Technology Licensing, LLC**
Redmond, WA 98052-6399 (US)

(72) Inventors:
- **WEBER, Jason**
  Redmond, Washington 98052-6399 (US)
- **TITUS, Tobin**
  Redmond, Washington 98052-6399 (US)
- **LIBBY, Daniel**
  Redmond, Washington 98052-6399 (US)
- **MANTHOS, Brian**
  Redmond, Washington 98052-6399 (US)
- **PACITTI, Colin**
  Redmond, Washington 98052-6399 (US)
- **ZHAO, Pengxiang**
  Redmond, Washington 98052-6399 (US)
- **MILLER, Matthew**
  Redmond, Washington 98052-6399 (US)
- **RABET, Jordan Thomas**
  Redmond, Washington 98052-6399 (US)
- **HAZEN, John**
  Redmond, Washington 98052-6399 (US)

(74) Representative: **CMS Cameron McKenna Nabarro Olswang LLP**
Cannon Place
78 Cannon Street
London EC4N 6AF (GB)

(56) References cited:
WO-A2-2013/172913 **WO-A2-2013/172913**
US-A1- 2014 380 474 **US-A1- 2014 380 474**

- **ASLAN ASKAROV ET AL: "Predictive black-box mitigation of timing channels", PROCEEDINGS OF THE 17TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 4 October 2010 (2010-10-04), pages 297-307, XP058270201, DOI: 10.1145/1866307.1866341 ISBN: 978-1-4503-0245-6**
- **ASLAN ASKAROV ET AL: "Predictive black-box mitigation of timing channels", PROCEEDINGS OF THE 17TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 4 October 2010 (2010-10-04), pages 297-307, XP058270201, DOI: 10.1145/1866307.1866341 ISBN: 978-1-4503-0245-6**

Note: Within nine months of the publication of the mention of the grant of the European patent in the European Patent Bulletin, any person may give notice to the European Patent Office of opposition to that patent, in accordance with the Implementing Regulations. Notice of opposition shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

EP 3 785 157 B1

Description

BACKGROUND

[0001] In computing, a timing attack is a type of side-channel attack (i.e., an attack based on information gained from the physical implementation of a computer system) in which an attacker attempts to compromise the system by analyzing the amount of time it takes to complete one or more operations. Every logical operation in a computer system takes some time to execute, and that time can differ based on the input(s) to the operation. Accordingly, with sufficiently precise measurements of an operation's execution time, an attacker can create a time model for the operation and deduce its input(s) (which may include a secret). Recent highly publicized security vulnerabilities that rely on timing attacks include the Meltdown and Spectre vulnerabilities which affect most modern microprocessor architectures.

[0002] Generally speaking, to carry out a timing attack, an attacker needs to be able to quantify an operation's execution time via a reference clock-in other words, a clock in which clock ticks arrive at a consistent rate. The attacker may establish the reference clock by consulting an explicit clock (i.e., one that is derived from hardware signals and typically represented in either wall clock time or CPU time). For example, the attacker may call an application programming interface (API) that returns timestamp values as determined by the system hardware. The attacker may also establish the reference clock by creating an implicit clock (i.e., one that is derived from an arbitrary unit of time measure, without need for an explicit clock). For example, the attacker may track the number of times it can take some action or carry out some task (e.g., call an API, run a calculation, etc.) while an operation executes and use that number to quantify the duration of the operation.

[0003] There are a number of existing approaches for mitigating timing attacks, such as clamping explicit clocks to a relatively low resolution, altering runtime hardware frequencies, and adding noise or interference to such frequencies. However, these existing approaches can be worked around and thus fail to structurally prevent timing exploits. Additionally, since these existing approaches cause a computer system to deviate from ideal operating conditions, they negatively impact the experience of users interacting with the system (e.g., performance becomes worse, resource usage increases, power efficiency decreases, etc.).

WO 2013/172913 A2 describes devices, systems, apparatus, methods, products, and other implementations, for identifying a process to obtain timing information of a processor-based device, and in response to identifying the process to obtain the timing information, delaying delivery of the timing information for a time-delay period. In some embodiments, identifying the process to obtain the timing information may include identifying a request to obtain the timing information of the processor-based device. In some embodiments, identifying the process to obtain the timing information may include identifying a memory-access process.

US 2014/380474 A1 describes a system comprises one or more counters; comparison logic; and one or more hardware processors communicatively coupled to the one or more counters and the comparison logic. The one or more hardware processors are configured to instantiate one or more virtual machines that are adapted to analyze received content, where the one or more virtual machines are configured to monitor a delay caused by one or more events conducted during processing of the content and identify the content as including malware if the delay exceed a first time period.

ASLAN ASKAROV ET AL, "Predictive black-box mitigation of timing channels", PROCEEDINGS OF THE 17TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, ACM, 2 PENN PLAZA, SUITE 701 NEW YORK NY 10121-0701 USA, 4 October 2010 describes techniques for general black-box mitigation of timing channels. The source of events is wrapped by a timing mitigator that delays output events so that they contain only a bounded amount of information. We introduce a general class of timing mitigators that can achieve any given bound on timing channel leakage, with a trade-off in system performance. We show these mitigators compose well with other mechanisms for information flow control, and demonstrate they are effective against some known timing attacks.

SUMMARY

[0004] The invention is set out in the appended set of claims.

[0005] Techniques for mitigating timing attacks via dynamically scaled time dilation are provided. According to one set of embodiments, a computer system can enable time dilation with respect to a program, where the time dilation causes the program to observe a dilated view of time relative to real time. Then, while the time dilation is enabled, the computer system can track a count of API calls or callbacks made by a program within each of a series of time buckets and, based on counts tracked for a range of recent time buckets, scale up or scale down a degree of the time dilation.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006]

FIG. 1 depicts a simplified block diagram of a software environment that implements the techniques of the present disclosure.

FIG. 2 depicts a workflow for dynamically triggering time dilation according to certain embodiments.

FIGS. 3A, 3B, and 3C depict graphs illustrating different algorithms for warping an explicit clock according to certain embodiments.

FIG. 4 depicts a workflow for dynamically scaling time dilation according to certain embodiments.
FIG. 5 depicts a simplified block diagram of a computer system according to certain embodiments.

DETAILED DESCRIPTION

[0007] In the following description, for purposes of explanation, numerous examples and details are set forth in order to provide an understanding of various embodiments. It will be evident, however, to one skilled in the art that certain embodiments can be practiced without some of these details, or can be practiced with modifications or equivalents thereof.

1. Overview

[0008] Embodiments of the present disclosure provide techniques for mitigating timing attacks via "time dilation"-in other words, dilating or warping the view of time of an observer (e.g., a potential attacker) relative to real time such that the observer cannot establish a consistent reference clock for carrying out a timing attack.

[0009] According to a first set of embodiments, a computer system can track the number of API calls and/or callbacks made by an observer within each of a series of time windows, referred to as buckets. The API calls/callbacks may be explicit clock APIs and/or APIs that can be used to construct implicit clocks. If the system determines that the observer has exceeded a threshold number of API calls/callbacks for a predefined consecutive number of buckets, the system can dynamic trigger (i.e., turn-on) time dilation with respect to that observer. This can include, e.g., (1) injecting a random amount of wait time (i.e., pausing) into the implicit clock API calls/callbacks called by the observer, thereby preventing the observer from constructing a consistent implicit clock from those calls/callbacks, and/or (2) randomly jittering/warping the time values returned by explicit clock APIs to the observer, thereby preventing the observer from using those explicit clock APIs establish a consistent view of real time.

[0010] According to a second set of embodiments, the computer system can dynamically scale the amount of time dilation that is introduced via (1) and (2) for the observer based on the observer's continued behavior/activity. For example, if the observer continues to act "badly" (e.g., issue a high number of API calls/callbacks over an extended period of time) which indicates that the observer is likely perpetrating a timing attack or continuing to perpetrate such an attack, the system may increase the amount of wait time injected into implicit clock API calls/callbacks called by the observer and/or increase the amount of jitter/warping of time values returned by explicit clock APIs to the observer. Conversely, if the observer's behavior improves (e.g., reduces its call/callback activity to a normal level for an extended period of time), the time dilation introduced via the wait time injection and explicit clock jittering/warping may be dialed back or even turned off entirely.

[0011] With the two high-level concepts described above (dynamic triggering and dynamic scaling of time dilation), the observer can run under ideal conditions, with no significant delay relative to real time, as long as the observer is well-behaved. However, as soon as the observer begins to exhibit behavior that is indicative of a timing attack, the system can introduce time dilation safeguards that prevent the observer from establishing a consistent reference clock via either implicit clocks or explicit clocks. Further, as the observer becomes more or less aggressive in its activity/behavior, the system can dynamically increase or decrease the degree of time dilation as appropriate. In this way, these techniques can mitigate timing attacks in a manner that is more intelligent, efficient, and performant than existing solutions.

[0012] The foregoing and other aspects of the present disclosure are described in further detail in the sections that follow. For purposes of illustration, the embodiments and examples below are presented in the context of a web platform application (e.g., a web browser) that runs JavaScript code downloaded by the application. In these examples/embodiments, the JavaScript code is the observer/potential perpetrator of timing attacks and the web platform application is the entity that implements the techniques of the present disclosure. This is a valuable use case to consider, since JavaScript and web content in general is typically the most common vector by which unknown, potentially malicious code can reach end-user systems. However, it should be appreciated the present disclosure is not solely limited to this context and can be applied to other contexts in which timing attack mitigations are useful and appropriate. By way of example, the techniques described herein are equally applicable for mitigating timing attacks in systemlevel software.

2. Software Environment

[0013] FIG. 1 is a simplified block diagram of a software environment 100 in which embodiments of the present disclosure may be implemented. As shown, software environment 100 includes a web platform application 102, which may be a web browser or any other software application that hosts/presents web content, and a JavaScript program 104 that is downloaded and run by application 102. Web platform application 102 includes a set of web platform APIs 106 that can be invoked by JavaScript program 104. Examples of such APIs include standard JavaScript APIs (including callback functions such as setTimeout(), setInterval(), etc.), Document Object Model (DOM) APIs, HTML5 APIs, browser-specific APIs, and so on.

[0014] Web platform application 102 also includes an event loop 108 that adheres to the HTML5 event loop standard and is generally responsible for coordinating the execution of different processes within web platform application 102, including JavaScript program 104. In

one set of embodiments, event loop 108 can execute code each time a given API 106 is invoked by JavaScript program 104 and thus can act as a "chokepoint" for all such API invocations.

[0015] As alluded to in the Background section, the mitigation of timing attacks is becoming an increasingly important issue with the emergence of far-reaching security vulnerabilities such as Meltdown and Spectre. JavaScript code, such as JavaScript program 104 of FIG. 1, is perhaps one of the most common vectors by which malicious code capable of perpetrating a timing attack can find its way onto end-user systems. For instance, JavaScript program 104 may be surreptitiously injected into an online advertisement that is then disseminated across a number of different websites via an ad network. Accordingly, it is desirable to have mitigations at the web platform level that prevent or make it difficult for JavaScript program 104 to carry out a timing attack.

[0016] Unfortunately, existing approaches to timing attack mitigation suffer from various limitations and disadvantages that make them less than ideal solutions for this use case. For examples, approaches that simply clamp explicit clocks to a coarse granularity or modify runtime frequencies introduce performance and power management problems that negatively affect the user experience of end-users interacting with the web platform (e.g., animations begin to stutter, UI responsiveness degrades, web pages are slow to load, etc.). These problems are particularly acute on mobile devices which depend on efficient resource usage and power management for day-long operation.

[0017] To address the foregoing and other similar issues, web platform application 102 of FIG. 1 is enhanced to include, within event loop 108, a call rate tracking module 110, a pause task module 112, and an explicit clock warp module 114. At a high level, modules 110-114 can interoperate to implement two novel timing attack mitigation mechanisms according to embodiments of the present disclosure: (1) dynamically triggered time dilation and (2) dynamically scaled time dilation.

[0018] With respect to mechanism (1) (described in further detail in section (3) below), call rate tracking module 110 can track the number of calls made by JavaScript program 104 to web platform APIs 106 that enable program 104 to construct an implicit clock (e.g., callback functions) and/or consult an explicit clock (e.g., timestamp APIs). Note that this is possible because event loop 108 acts as a chokepoint for all of the API invocations made by JavaScript program 104 and other processes of web platform application 102. Call rate tracking module 110 can perform this tracking on a per-bucket basis, where each bucket is a time window of a predefined period (e.g., 200 milliseconds), and can compare the number of calls/callbacks made within each bucket to a threshold. If call rate tracking module 110 determines that the threshold has been exceeded for a certain number of consecutive buckets, event loop 108 can trigger time dilation for JavaScript program 104 by inserting wait time into each implicit clock API call/callback via pause task module 112, and/or warping the time values returned by explicit clock APIs via explicit clock warp module 114. The end result of this is that JavaScript program 104 begins to observe a dilated view of time that is inconsistent with real time, and thus makes it difficult or impossible for program 104 to construct a consistent reference clock in order to perpetrate a timing attack.

[0019] Significantly, since the wait time insertion and explicit clock warping is only turned-on in scenarios where JavaScript program 104 is deemed to be a potential attacker (via the bucket-based call tracking above), this approach does not introduce any performance or resource usage overhead for web content that is well-behaved. This is a significant advantage over existing mitigation techniques, which tend to turn-on heavyhanded mitigations by default and thus introduce performance/power problems for all web pages, whether good or bad.

[0020] With respect to mechanism (2) (described in further detail in section (5) below), call rate tracking module 110 can continue to track the number of calls/callbacks made by JavaScript program 104 to web platform APIs 106 on a per-bucket basis once time dilation is turned on (either via the dynamic triggering mechanism of (1) or via a static configuration). Based on this continued tracking, call rate tracking module 110 can apply one or more policies to determine whether JavaScript program 104 is becoming more or less aggressive in its API calling behavior (indicating that the program is likely continuing to, or is no longer or perhaps never was, attempting to perpetrate a timing attack). This, in turn, can cause the system to scale up or down the degree of time dilation for JavaScript program 104 (via pause task module 112 and explicit clock warp module 114) in a proportional way. For example, if JavaScript program 104 continues to call implicit or explicit clock-related APIs at a high frequency for an extended period of time, the system can conclude that program 104 is continuing to perpetrate a timing attack and can ramp up the amount of wait time inserted into each implicit clock-related API call/callback, and/or the amount of warping applied to explicit clock time values. This ramping-up process can continue as long as JavaScript program 104 persists in its bad behavior, and may ultimately cause program 104 to be terminated.

[0021] Conversely, if the API call/callback rate of JavaScript program 104 drops to a low or normal level for an extended period of time, event loop 108 can conclude that program 104 is now well-behaved and can begin ramping down the amount of wait time inserted into each implicit clock-related API call/callback, and/or the amount of warping applied to explicit clock time values. This ramping-down process can continue as long as JavaScript program 104 persists in its good behavior, and may ultimately cause time dilation to be turned off entirely for program 104.

[0022] Thus, with mechanism (2), web platform application 102 can more intelligently apply its timing attack

mitigations in a manner that is proportional and responsive to the real-time activity/behavior of JavaScript program 104.

[0023] It should be appreciated that software environment 100 of FIG. 1 is illustrative and not intended to limit embodiments of the present disclosure. For example, while the various entities shown in this figure are arranged according to a particular configuration, other configurations are also possible. Further, these entities may include various subcomponents and/or functions that are not specifically described. One of ordinary skill in the art will recognize other variations, modifications, and alternatives.

3. Dynamically Triggering Time Dilation

[0024] FIG. 2 depicts a workflow 200 that provides additional details regarding the processing that may be performed by web platform application 102 and its constituent components (e.g., call rate tracking module 110, pause task module 112, explicit clock warp module 114) for dynamically triggering time dilation with respect to JavaScript program 104 according to certain embodiments.

[0025] At block 202, while JavaScript program 104 is running, call rate tracking module 110 can start a timer for a current bucket (i.e., time window). At block 204, call rate tracking module 110 can receive and count the number of calls/callbacks made by JavaScript program 104 to web platform APIs 106 that either enable program 104 to construct an implicit clock (i.e., a clock based on an arbitrary, internal unit of time measure) or consult an explicit clock (i.e., a clock that is based on hardware signals and represented as, e.g., wall clock time or CPU time). Examples of the former include JavaScript API calls or callbacks such as setTimeout() and setIntervalU. Examples of the latter include any API that returns a hardware-derived timestamp or time value.

[0026] At block 206, call rate tracking module 110 can determine that the timer started at block 202 has reached a predefined time limit (e.g., 200 milliseconds) and close the current bucket. In addition, module 110 can record the total number of API calls/callbacks counted during that bucket (block 208) and check whether the total number exceeds a predefined threshold (block 210). If so, call rate tracking module 110 can mark the bucket as a "bad" bucket (block 212). Otherwise, call rate tracking module 110 can mark the bucket as a "good" bucket (block 214).

[0027] Once call rate tracking module 110 has marked the bucket appropriately, module 110 can check whether the last X consecutive buckets were bad buckets, where X is some predefined number (block 216). If not, call rate tracking module 110 can return to block 202 in order to start a timer for a new bucket and repeat the preceding steps.

[0028] However, if the last X consecutive buckets were in fact bad buckets, it can be concluded that JavaScript program 104 is exhibiting bad behavior that is indicative of a timing attack. As a result, web platform application 102 can trigger (i.e., turn-on) time dilation with respect to JavaScript program 104 by leveraging pause task module 112 and/or explicit clock warp module 114 (block 218).

[0029] For example, according to one set of embodiments, for each successive call that JavaScript program 104 makes to an API function or callback that relates to implicit clock creation, event loop 108 can (via, e.g., a task scheduler) instruct pause task module 112 to insert a randomly generated amount of wait time into the API execution flow, before the call/callback returns to program 104. The result of this is that the API never completes in a consistent amount of time from the perspective of JavaScript program 104, which makes it difficult or impossible for program 104 to count instances of these API calls/callbacks to construct an implicit clock. In a particular embodiment, the amount of wait time that pause task module 112 inserts into each API call/callback can be a random value from 0 to 255 microseconds.

[0030] According to another set of embodiments, for each successive call that JavaScript program 104 makes to an API function that relates to an explicit clock, event loop 108 can (via, e.g., a task scheduler) instruct explicit clock warp module 114 to randomly dilate or warp the time value that is returned by the API to program 104. The result of this is that JavaScript program 104 never receives a consistent view of time from these explicit clock APIs, which makes it difficult or impossible for program 104 to create a consistent reference clock based on the explicit clocks. There are different ways in which explicit clock warp module 114 can warp the time values that are generated by the explicit clock APIs, which include clamping/random jitter and applying randomly-generated linear or higher-order functions that transform real time to warped time. These various techniques are discussed in section (4) below.

[0031] According to yet other embodiments, event loop 108 can trigger any combination or subset of the time dilation techniques described above according to various policies. For example, if call rate tracking module 110 determines that JavaScript program 104 has invoked a threshold number of explicit clock APIs for X consecutive bad buckets (but not a threshold number of APIs related to implicit clocks), event loop 108 may solely trigger explicit clock warping. As another example, if call rate tracking module 110 determines that JavaScript program 104 has invoked a threshold number of APIs related to implicit clocks for X consecutive bad buckets (but not a threshold number of explicit clock APIs), event loop 108 may solely trigger wait time insertion. As yet another example, a large number of API calls/callbacks for implicit clocks may also trigger explicit clock warping, and vice versa. All of these permutations, and more, are within the scope of the present disclosure.

[0032] It should be appreciated that workflow 200 of FIG. 2 is illustrative and various modifications are possi-

ble. For example, although workflow 200 shows that a single instance of call rate tracker module 110 can track the counts of both explicit and implicit clock API calls/callbacks, in some cases two separate instances of module 110 may be used for these purposes respectively. Further, the various steps shown in workflow 200 can be sequenced differently, certain steps can be combined as needed, and certain steps can be omitted as needed. One of ordinary skill in the art will recognize other variations, modifications, and alternatives.

4. Warping Explicit Clocks

[0033] As mentioned above, at the time of determining that time dilation should be turned on with respect to JavaScript program 104, web platform application 102 can leverage explicit clock warp module 114 in order to dilate or warp the time values that are returned by explicit clocks APIs to program 104, thereby preventing program 104 from observing a consistent view of time via explicit clocks. Generally speaking, explicit clock warp module 114 can use any algorithm to transform the time values returned by the explicit clock APIs (referred to as real time) into the time values observed by JavaScript program 104 (referred to as observed time), as long as program 104's observed view of time is non-decreasing.

[0034] According to one set of embodiments, explicit clock warp module 114 can perform this warping by clamping the time values to a relatively coarse granularity, such as 5 or 20 microsecond intervals, and then randomly jittering the point at which a particular time value is clamped (i.e., performing the clamping at different random times within each clamping period). These concepts are visualized in FIGS. 3A and 3B according to an embodiment. In particular, FIG. 3A depicts a graph 300 with real time on the x-axis and observed time on the y-axis, where the y values (i.e., time observed by JavaScript program 104) are clamped at regular 20 microsecond intervals. FIG. 3B depicts a graph 310 where the y values are clamped, but the point at which the clamping occurs is randomly jittered. This results in clamping periods of different random lengths.

[0035] According to another set of embodiments, explicit clock warp module 114 can perform the warping by using a linear transformation function such as $y = ax + b$ where variables a and b are chosen randomly, or a non-linear transformation function such as $y = ax^t + b$ where variables a, b, and t are chosen randomly. An example nonlinear transformation function is shown as graph 320 in FIG. 3C. One advantage of using these transformation functions over the random jitter technique is that the delay experienced by JavaScript program 104 is less variable; in the case of the transformation functions, the maximum delay will be defined by the function itself, whereas with random jittering the maximum delay may be as high as twice the clamping interval (depending upon how the clamping turnover points are randomly determined).

[0036] In various embodiments, the transformation function described above can be made as complex as needed (by, e.g., adding more variables/dimensions) in order to make it difficult for an attacker to reverse-engineer the function and determine how time is being warped. In some embodiments, multiple transformation functions may be spliced together for further security.

5. Dynamically Scaling Time Dilation

[0037] FIG. 4 depicts a workflow 400 that provides additional details regarding the processing that may be performed by web platform application 102 and its constituent components (e.g., call rate tracking module 110, pause task module 112, explicit clock warp module 114) for dynamically scaling time dilation with respect to JavaScript program 104 according to certain embodiments. Workflow 400 assumes that time dilation has already been turned on for JavaScript program 104, either by virtue of the dynamic triggering mechanism described in section (3) or via a static (e.g., default) configuration.

[0038] Blocks 402-414 of workflow 400 are substantially similar to blocks 302-314 of workflow 300. In particular, at block 402, call rate tracking module 110 can start a timer for a current bucket (i.e., time window). While this timer is running, call rate tracking module 110 can receive and count the number of calls/callbacks made by JavaScript program 104 to web platform APIs 106 that either enable program 104 to construct an implicit clock (i.e., a clock based on an arbitrary, internal unit of time measure) or consult an explicit clock (i.e., a clock that is based on hardware signals and represented as, e.g., wall clock time or CPU time) (block 404).

[0039] At block 406, call rate tracking module 110 can determine that the timer started at block 402 has reached a predefined time limit (e.g., 200 milliseconds) and close the current bucket. In addition, module 110 can record the total number of API calls/callbacks counted during that bucket (block 408) and check whether that total number exceeds a predefined threshold (block 410). If so, call rate tracking module 110 can mark the bucket as a bad bucket (block 412). Otherwise, call rate tracking module 110 can mark the bucket as a good bucket (block 414).

[0040] Once call rate tracking module 110 has marked the bucket appropriately, module 110 can check the number of bad buckets have been encountered within some range of Y recent buckets (block 416). Note that this condition is different from that used in the triggering workflow (which looks at consecutive bad buckets), since when scaling time dilation it is generally more useful to look at patterns of behavior over non-contiguous periods of time (to account for scenarios where JavaScript program 104 may temporarily halt or slow down its call rate activity in an attempt to fool mitigation mechanisms).

[0041] If the number of bad buckets encountered within the last Y buckets is between some low watermark A and some high watermark B, it can be concluded that the call rate behavior of JavaScript program 104 is about the

same as before (i.e., has gotten neither better nor worse) and call rate tracking module 110 can return to block 402 in order to start a timer for a new bucket and repeat the preceding steps. Note that in this case, pause task module 112 and explicit clock warp module 114 will continue to insert wait time and warp explicit clock values for JavaScript program 104 in accordance with what they were doing before.

[0042] On the other hand, if the number of bad buckets encountered within the last Y buckets is greater than the high watermark B, it can be concluded that the call rate behavior/activity of JavaScript program 104 is increasing/getting worse. In this case, high watermark B can be incremented/increased and web platform application 102 can scale up the degree of time dilation applied to JavaScript program 104 (block 418). For example, if pause task module 112 was previously inserting a randomly generated amount of wait time into the API execution flow according to a certain range (e.g., 0 to 255 microseconds), module 112 can increase the top value of this range such that the maximum possible wait time is increased. Further, if explicit clock warp module 114 was previously warping the time values returned by explicit clock APIs to program 104 according to some clamping interval and some amount of random jitter, module 114 can increase the clamping interval and/or range of random jitter, such that the time observed by JavaScript program 104 is even further removed from real time. Call rate tracking module 110 can then return to block 402 in order to start a timer for a new bucket and repeat the preceding steps.

[0043] Finally, if the number of bad buckets encountered within the last Y buckets is less than the low watermark A, it can be concluded that the call rate behavior/activity of JavaScript program 104 is decreasing/getting better. In this case, low watermark A can be decremented/decreased and web platform application 102 can scale down the degree of time dilation applied to JavaScript program 104 (block 420). For example, if pause task module 112 was previously inserting a randomly generated amount of wait time into the API execution flow according to a certain range, module 112 can decrease the top value of this range such that the maximum possible wait time is decreased. Further, if explicit clock warp module 114 was previously warping the time values returned by explicit clock APIs to program 104 according to some clamping interval and some amount of random jitter, module 114 can decrease the clamping interval and/or range of random jitter, such that the time observed by JavaScript program 104 is less removed from real time. Call rate tracking module 110 can then return to block 402 in order to start a timer for a new bucket and repeat the preceding steps.

[0044] Generally speaking, in various embodiments, web platform application 102 can scale up or down the wait time inserted by pause task module 112 and warping performed by explicit clock warp module 114 either independently or in a combined manner according to various policies. For example, if call rate tracking module 110 detects a significant change in call rate behavior with respect to explicit clock APIs but not implicit clock APIs, application 102 may solely scale explicit clock warping. Conversely, if call rate tracking module 110 detects a significant change in call rate behavior with respect to implicit clock APIs but not explicit clock APIs, application 102 may solely scale wait time insertion. As another example, a significant change in behavior with respect to implicit clocks may also cause scaling of explicit clock warping, and vice versa. All of these permutations, and more, are within the scope of the present disclosure.

[0045] Further, although not shown in FIG. 4, in some embodiments web platform application 102 may take special measures if JavaScript program 104 behaves exceptionally well or badly over an extended period of time. For instance, if high watermark B reaches a maximum threshold (indicating that the call rate activity/behavior of JavaScript program 104 has gotten progressively worse), application 102 may simply terminate the process associated with the program rather than continuing to scale the time dilation experienced by the program any further. Similarly, if low watermark A reaches a minimum threshold (indicating that the call rate activity/behavior of JavaScript program 104 has gotten progressively better), application 102 may completely turn off time dilation for program 104. In this scenario, time dilation may be triggered again per workflow 300 if JavaScript program 104 begins exhibiting bad behavior again in the future.

6. Leveraging Telemetry

[0046] In some embodiments, web platform application 102 can include a telemetry component that enables it to communicate information regarding the time dilation performed with respect to JavaScript program 104 and other downloaded JavaScript code to one or more remote servers. Examples of such information include the web pages/URLs comprising program 104, the measured call rate activity of program 104, whether time dilation was triggered for program 104, whether time dilation was scaled for program 104, the particular parameters used for the time dilation triggering/scaling, system performance parameters when time dilation was turned on, and so on.

[0047] The servers can then aggregate this information across a large population of applications/users and determine statistics and trends which can be used for various purposes. For example, in one set of embodiments, the servers can identify a "whitelist" of web pages that are highly unlikely to contain JavaScript code that is attempting to perpetrate a timing attack, as well as a "blacklist" of web pages that are highly likely to contain such malicious JavaScript code. The whitelist and blacklist can then be communicated back to web platform application 102 (and/or to other client applications such as anti-virus/anti-malware software), which can uses these lists to, e.g., block user access to blacklisted sites, turn off

time dilation by default for whitelisted sites, and/or implement more relaxed time dilation policies/parameters/rules for whitelisted sites.

[0048] As another example, the statistics determined by the remote servers can be used to inform and fine tune the time dilation triggering and scaling algorithms described above, such that they perform as expected and without unnecessary or burdensome performance penalties. For instance, the remote servers may determine that one particular algorithm used to perform explicit clock warping results in excessive stuttering on a few popular websites, which may instigate a change to a different algorithm. Further, the remote servers may determine that the scaling algorithm may cause a particular nonmalicious website to crash when loaded, which may result in a modification to the scaling parameters to avoid this incompatibility. One of ordinary skill in the art will recognize other possible use cases for this collected data.

7. Example Computer System

[0049] FIG. 5 is a simplified block diagram illustrating the architecture of an example computer system 500 according to certain embodiments. Computer system 500 (and/or equivalent systems/devices) may be used to run any of the software described in the foregoing disclosure, including web platform application 102 of FIG. 1. As shown in FIG. 5, computer system 500 includes one or more processors 502 that communicate with a number of peripheral devices via a bus subsystem 504. These peripheral devices include a storage subsystem 506 (comprising a memory subsystem 508 and a file storage subsystem 510), user interface input devices 512, user interface output devices 514, and a network interface subsystem 516.

[0050] Bus subsystem 504 can provide a mechanism for letting the various components and subsystems of computer system 500 communicate with each other as intended. Although bus subsystem 504 is shown schematically as a single bus, alternative embodiments of the bus subsystem can utilize multiple busses.

[0051] Network interface subsystem 516 can serve as an interface for communicating data between computer system 500 and other computer systems or networks. Embodiments of network interface subsystem 516 can include, e.g., an Ethernet module, a Wi-Fi and/or cellular connectivity module, and/or the like.

[0052] User interface input devices 512 can include a keyboard, pointing devices (e.g., mouse, trackball, touchpad, etc.), a touch-screen incorporated into a display, audio input devices (e.g., voice recognition systems, microphones, etc.), motion-based controllers, and other types of input devices. In general, use of the term "input device" is intended to include all possible types of devices and mechanisms for inputting information into computer system 500.

[0053] User interface output devices 514 can include a display subsystem and nonvisual output devices such as audio output devices, etc. The display subsystem can be, e.g., a transparent or non-transparent display screen such as a liquid crystal display (LCD) or organic light-emitting diode (OLED) display that is capable of presenting 2D and/or 3D imagery. In general, use of the term "output device" is intended to include all possible types of devices and mechanisms for outputting information from computer system 500.

[0054] Storage subsystem 506 includes a memory subsystem 508 and a file/disk storage subsystem 510. Subsystems 508 and 510 represent non-transitory computer-readable storage media that can store program code and/or data that provide the functionality of embodiments of the present disclosure.

[0055] Memory subsystem 508 includes a number of memories including a main random access memory (RAM) 518 for storage of instructions and data during program execution and a read-only memory (ROM) 520 in which fixed instructions are stored. File storage subsystem 510 can provide persistent (i.e., non-volatile) storage for program and data files, and can include a magnetic or solid-state hard disk drive, an optical drive along with associated removable media (e.g., CD-ROM, DVD, Blu-Ray, etc.), a removable or non-removable flash memory-based drive, and/or other types of storage media known in the art.

[0056] It should be appreciated that computer system 500 is illustrative and other configurations having more or fewer components than computer system 500 are possible.

[0057] The above description illustrates various embodiments of the present disclosure along with examples of how aspects of these embodiments may be implemented. The above examples and embodiments should not be deemed to be the only embodiments, and are presented to illustrate the flexibility and advantages of the present disclosure as defined by the following claims. For example, although certain embodiments have been described with respect to particular process flows and steps, it should be apparent to those skilled in the art that the scope of the present disclosure is not strictly limited to the described flows and steps. Steps described as sequential may be executed in parallel, order of steps may be varied, and steps may be modified, combined, added, or omitted. As another example, although certain embodiments have been described using a particular combination of hardware and software, it should be recognized that other combinations of hardware and software are possible, and that specific operations described as being implemented in software can also be implemented in hardware and vice versa.

[0058] The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense. Other arrangements, embodiments, implementations and equivalents will be evident to those skilled in the art and may be employed without departing from the scope of the present disclosure as set forth in the follow-

ing claims.

Claims

1. A computer system (500) comprising:

   a processor (502); and
   a computer readable storage medium (508, 510) having stored thereon program code that, when executed by the processor, causes the processor to:

   enable time dilation with respect to a program, the time dilation causing the program to observe a dilated view of time relative to real time (218); and
   while the time dilation is enabled:

   track a count of application programming interface (API) calls or callbacks made by a program within each of a series of time buckets (404); and
   based on counts tracked for a range of recent time buckets, scale up or scale down a degree of the time dilation (418, 420),

   wherein if the count exceeds a threshold count within each of a high watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled up (416, 418), and/or wherein if the count falls below a threshold count within each of a low watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled down (416, 420).

2. The computer system of claim 1 wherein the count of API calls or callbacks include one or more calls or callbacks to APIs usable by the program for constructing an implicit clock (404).

3. The computer system of claim 2 wherein scaling up or scaling down the degree of the time dilation comprises:
   increasing or decreasing an amount of wait time inserted into the calls or callbacks to the APIs usable for constructing an implicit clock (112).

4. The computer system of claim 1 wherein the count of API calls or callbacks include one or more calls to explicit clock APIs configured to return time values derived from one or more hardware signals of the computer system (404).

5. The computer system of claim 4 wherein scaling up or scaling down the degree of the time dilation comprises: increasing or decreasing a degree of warping applied to the time values returned by the explicit clock APIs before passing the time values to the program, wherein the warping is performed by transforming the time values using a randomly determined linear or non-linear transformation function (114, 320).

6. A method for mitigating timing attacks via dynamically scaled time dilation, the method comprising:

   enabling, by a computer system, time dilation with respect to a program, the time dilation causing the program to observe a dilated view of time relative to real time (218); and
   while the time dilation is enabled:

   tracking, by the computer system, a count of application programming interface (API) calls or callbacks made by a program within each of a series of time buckets (404); and based on counts tracked for a range of recent time buckets, scaling up or scaling down a degree of the time dilation (418, 420),

   wherein if the count exceeds a threshold count within each of a high watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled up (416, 418), and/or wherein if the count falls below a threshold count within each of a low watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled down (416, 420).

7. The method of claim 6 wherein the count of API calls or callbacks include one or more calls or callbacks to APIs usable by the program for constructing an implicit clock (404).

8. The method of claim 6 wherein the count of API calls or callbacks include one or more calls to explicit clock APIs configured to return time values derived from one or more hardware signals of the computer system (404).

9. A computer readable storage medium having stored thereon program code executable by a computer system, the program code causing the computer system to:

   enable time dilation with respect to a program, the time dilation causing the program to observe a dilated view of time relative to real time (218); and
   while the time dilation is enabled:

   track a count of application programming interface (API) calls or callbacks made by a

program within each of a series of time buckets (404); and
based on counts tracked for a range of recent time buckets, scale up or scale down a degree of the time dilation (418, 420),

wherein if the count exceeds a threshold count within each of a high watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled up (416, 418), and/or wherein if the count falls below a threshold count within each of a low watermark of time buckets in the range of recent time buckets, the degree of the time dilation is scaled down (416, 420).

10. The computer readable storage medium of claim 9 wherein the count of API calls or callbacks include one or more calls or callbacks to APIs usable by the program for constructing an implicit clock (404).

11. The computer readable storage medium of claim 9 wherein the count of API calls or callbacks include one or more calls to explicit clock APIs configured to return time values derived from one or more hardware signals of the computer system (404).

Patentansprüche

1. Rechensystem (500), umfassend:

einen Prozessor (502); und
ein computerlesbares Speichermedium (508, 510), auf dem Programmcode gespeichert ist, der, wenn er von dem Prozessor ausgeführt wird, den Prozessor zu Folgendem veranlasst:

Aktivieren der Zeitdilatation in Bezug auf ein Programm, wobei die Zeitdilatation das Programm veranlasst, eine gedehnte Sicht der Zeit relativ zu der Echtzeit (218) zu beobachten; und
während die Zeitdilatation aktiviert ist:

Verfolgen einer Anzahl von Aufrufen der Anwendungsprogrammierschnittstelle (API) oder von Rückrufen, die von einem Programm innerhalb einer Reihe von Zeitabschnitten (404) gemacht wurden; und
basierend auf der verfolgten Anzahl für eine Reihe von Zeitabschnitten, Vergrößern oder Verkleinern eines Grades der Zeitdilatation (418, 420),

wobei, wenn die Anzahl einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte überschreitet, der Grad der Zeitdilatation vergrößert wird (416, 418), und/oder
wobei, wenn die Anzahl unter einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte fällt, der Grad der Zeitdilatation verkleinert wird (416, 420).

2. Rechensystem nach Anspruch 1, wobei die Anzahl der API-Aufrufe oder - Rückrufe einen oder mehrere Aufrufe oder Rückrufe zu APIs beinhaltet, die von dem Programm zur Konstruktion einer impliziten Uhr (404) verwendet werden können.

3. Rechensystem nach Anspruch 2, wobei das Vergrößern oder Verkleinern des Grades der Zeitdilatation umfasst:
Erhöhen oder Verringern eines Betrags an Wartezeit, der in die Aufrufe oder Rückrufe zu den APIs eingefügt wird, die für die Konstruktion einer impliziten Uhr (112) verwendet werden können.

4. Rechensystem nach Anspruch 1, wobei die Anzahl der API-Aufrufe oder Rückrufe einen oder mehrere Aufrufe zu expliziten Uhren-APIs beinhaltet, die so konfiguriert sind, dass sie Zeitwerte zurückgeben, die von einem oder mehreren Hardwaresignalen des Rechensystems (404) abgeleitet sind.

5. Rechensystem nach Anspruch 4, wobei das Vergrößern oder Verkleinern des Grades der Zeitdilatation umfasst:
Erhöhen oder Verringern eines Grades der Verzerrung, der auf die von den expliziten Uhren-APIs zurückgegebenen Zeitwerte angewandt wird, bevor die Zeitwerte an das Programm weitergegeben werden, wobei die Verzerrung durch Transformieren der Zeitwerte unter Verwendung einer zufällig bestimmten linearen oder nichtlinearen Transformationsfunktion (114, 320) durchgeführt wird.

6. Verfahren zum Abschwächen von Zeitangriffen mittels dynamisch skalierter Zeitdilatation, wobei das Verfahren umfasst:

Aktivieren der Zeitdilatation in Bezug auf ein Programm durch ein Rechensystem, wobei die Zeitdilatation das Programm veranlasst, eine gedehnte Sicht der Zeit relativ zu der Echtzeit (218) zu beobachten; und
während die Zeitdilatation aktiviert ist:

Verfolgen einer Anzahl von Aufrufen der Anwendungsprogrammierschnittstelle
(API) oder von Rückrufen durch das Rechensystem, die von einem Programm innerhalb einer Reihe von Zeitabschnitten (404) gemacht wurden; und

basierend auf der verfolgten Anzahl für eine Reihe von Zeitabschnitten, Vergrößern oder Verkleinern eines Grades der Zeitdilatation (418, 420),

wobei, wenn die Anzahl einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte überschreitet, der Grad der Zeitdilatation vergrößert wird (416, 418), und/oder
wobei, wenn die Anzahl unter einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte fällt, der Grad der Zeitdilatation verkleinert wird (416, 420).

7. Verfahren nach Anspruch 6, wobei die Anzahl der API-Aufrufe oder -Rückrufe einen oder mehrere Aufrufe oder Rückrufe zu APIs beinhaltet, die von dem Programm zur Konstruktion einer impliziten Uhr (404) verwendet werden können.

8. Verfahren nach Anspruch 6, wobei die Anzahl der API-Aufrufe oder Rückrufe einen oder mehrere Aufrufe zu expliziten Uhren-APIs beinhaltet, die so konfiguriert sind, dass sie Zeitwerte zurückgeben, die von einem oder mehreren Hardwaresignalen des Rechensystems (404) abgeleitet sind.

9. Computerlesbares Speichermedium, auf dem ein Programmcode gespeichert ist, der von einem Rechensystem ausgeführt werden kann, wobei der Programmcode das Rechensystem zu Folgendem veranlasst:

Aktivieren der Zeitdilatation in Bezug auf ein Programm, wobei die Zeitdilatation das Programm veranlasst, eine gedehnte Sicht der Zeit relativ zu der Echtzeit (218) zu beobachten; und während die Zeitdilatation aktiviert ist:

Verfolgen einer Anzahl von Aufrufen der Anwendungsprogrammierschnittstelle (API) oder von Rückrufen, die von einem Programm innerhalb einer Reihe von Zeitabschnitten (404) gemacht wurden; und basierend auf der verfolgten Anzahl für eine Reihe von Zeitabschnitten, Vergrößern oder Verkleinern eines Grades der Zeitdilatation (418, 420),

wobei, wenn die Anzahl einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte überschreitet, der Grad der Zeitdilatation vergrößert wird (416, 418), und/oder
wobei, wenn die Anzahl unter einen Schwellenwert innerhalb jedes einer hohen Wassermarke von Zeitabschnitten im Bereich der jüngsten Zeitabschnitte fällt, der Grad der Zeitdilatation verkleinert wird (416, 420).

10. Computerlesbares Speichermedium nach Anspruch 9, wobei die Anzahl der API-Aufrufe oder -Rückrufe einen oder mehrere Aufrufe oder Rückrufe zu APIs beinhaltet, die von dem Programm zur Konstruktion einer impliziten Uhr (404) verwendet werden können.

11. Computerlesbares Speichermedium nach Anspruch 9, wobei die Anzahl der API-Aufrufe oder Rückrufe einen oder mehrere Aufrufe zu expliziten Uhren-APIs beinhaltet, die so konfiguriert sind, dass sie Zeitwerte zurückgeben, die von einem oder mehreren Hardwaresignalen des Rechensystems (404) abgeleitet sind.

Revendications

1. Système informatique (500) comprenant :

un processeur (502) ; et
un support de stockage lisible par ordinateur (508, 510) sur lequel est stocké un code de programme qui, lorsqu'il est exécuté par le processeur, amène le processeur à :

activer une dilatation temporelle par rapport à un programme, la dilatation temporelle amenant le programme à observer une vue dilatée du temps par rapport à un temps réel (218) ; et
tandis que la dilatation temporelle est activée :

suivre un nombre d'appels ou de rappels à une interface de programmation d'application (API) réalisés par un programme au sein de chaque compartiment d'une série de compartiments temporels (404) ; et
en fonction des nombres suivis pour une plage de compartiments temporels récents, échelonner un degré de la dilatation temporelle à la hausse ou à la baisse (418, 420),

dans lequel, si le nombre dépasse un nombre seuil au sein de chaque repère haut de compartiments temporels dans la plage de compartiments temporels récents, le degré de la dilatation temporelle est échelonné à la hausse (416, 418), et/ou
dans lequel, si le nombre descend sous un nombre seuil au sein de chaque repère bas

de compartiments temporels dans la plage de compartiments temporels récents, le degré de la dilatation temporelle est échelonné à la baisse (416, 420).

2. Système informatique selon la revendication 1 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels ou rappels à des API utilisables par le programme pour construire une horloge implicite (404).

3. Système informatique selon la revendication 2 dans lequel l'échelonnement à la hausse ou à la baisse du degré de dilatation temporelle comprend :
l'augmentation ou la diminution d'une quantité de temps d'attente insérée dans les appels ou les rappels aux API utilisables pour construire une horloge implicite (112).

4. Système informatique selon la revendication 1 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels à des API d'horloge explicite configurées pour renvoyer des valeurs temporelles dérivées d'un ou plusieurs signaux matériels du système informatique (404).

5. Système informatique selon la revendication 4 dans lequel l'échelonnement à la hausse ou à la baisse du degré de dilatation temporelle comprend :
l'augmentation ou la diminution d'un degré de déformation appliqué aux valeurs temporelles renvoyées par les API d'horloge explicite avant de transférer les valeurs temporelles au programme, dans lequel la déformation est réalisée en transformant les valeurs temporelles à l'aide d'une fonction de transformation linéaire ou non linéaire déterminée de manière aléatoire (114, 320).

6. Procédé d'atténuation d'attaques temporelles via une dilatation temporelle échelonnée dynamiquement, le procédé comprenant :

l'activation, par un système informatique, d'une dilatation temporelle par rapport à un programme, la dilatation temporelle amenant le programme à observer une vue dilatée du temps par rapport à un temps réel (218) ; et
tandis que la dilatation temporelle est activée :

le suivi, par le système informatique, d'un nombre d'appels ou de rappels à une interface de programmation d'application (API) réalisés par un programme au sein de chaque compartiment d'une série de compartiments temporels (404) ; et
en fonction des nombres suivis pour une plage de compartiments temporels récents, l'échelonnement d'un degré de la dilatation temporelle à la hausse ou à la baisse (418, 420),
dans lequel, si le nombre dépasse un nombre seuil au sein de chaque repère haut de compartiments temporels dans la plage de compartiments temporels récents, le degré de la dilatation temporelle est échelonné à la hausse (416, 418), et/ou
dans lequel, si le nombre descend sous un nombre seuil au sein de chaque repère bas de compartiments temporels dans la plage de compartiments temporels récents, le degré de la dilatation temporelle est échelonné à la baisse (416, 420).

7. Procédé selon la revendication 6 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels ou rappels à des API utilisables par le programme pour construire une horloge implicite (404).

8. Procédé selon la revendication 6 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels à des API d'horloge explicite configurées pour renvoyer des valeurs temporelles dérivées d'un ou plusieurs signaux matériels du système informatique (404).

9. Support de stockage lisible par ordinateur sur lequel est stocké un code de programme exécutable par un système informatique, le code de programme amenant le système informatique à :

activer une dilatation temporelle par rapport à un programme, la dilatation temporelle amenant le programme à observer une vue dilatée du temps par rapport à un temps réel (218) ; et
tandis que la dilatation temporelle est activée :

suivre un nombre d'appels ou de rappels à une interface de programmation d'application (API) réalisés par un programme au sein de chaque compartiment d'une série de compartiments temporels (404) ; et
en fonction des nombres suivis pour une plage de compartiments temporels récents, échelonner un degré de la dilatation temporelle à la hausse ou à la baisse (418, 420),

dans lequel, si le nombre dépasse un nombre seuil au sein de chaque repère haut de compartiments temporels dans la plage de compartiments temporels récents, le degré de la dilatation temporelle est échelonné à la hausse (416, 418), et/ou
dans lequel, si le nombre descend sous un nombre seuil au sein de chaque repère bas de compartiments temporels dans la plage de compar-

timents temporels récents, le degré de la dilatation temporelle est échelonné à la baisse (416, 420).

10. Support de stockage lisible par ordinateur selon la revendication 9 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels ou rappels à des API utilisables par le programme pour construire une horloge implicite (404).

11. Support de stockage lisible par ordinateur selon la revendication 9 dans lequel le nombre d'appels ou de rappels à une API comprend un ou plusieurs appels à des API d'horloge explicite configurées pour renvoyer des valeurs temporelles dérivées d'un ou plusieurs signaux matériels du système informatique (404).

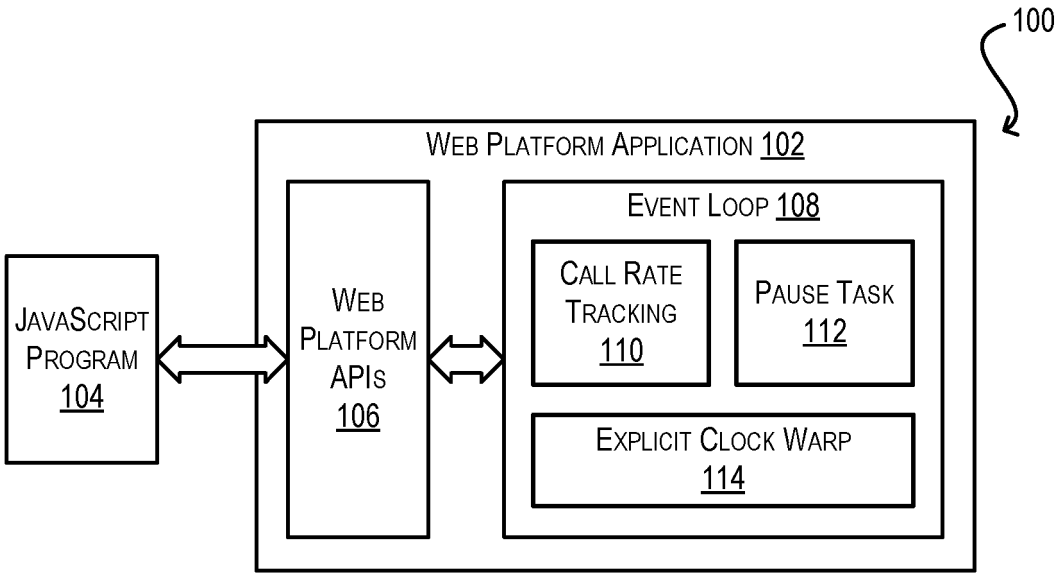
100
WEB PLATFORM APPLICATION 102
EVENT LOOP 108
CALL RATE TRACKING 110
PAUSE TASK 112
EXPLICIT CLOCK WARP 114
WEB PLATFORM APIS 106
JAVASCRIPT PROGRAM 104

FIG. 1

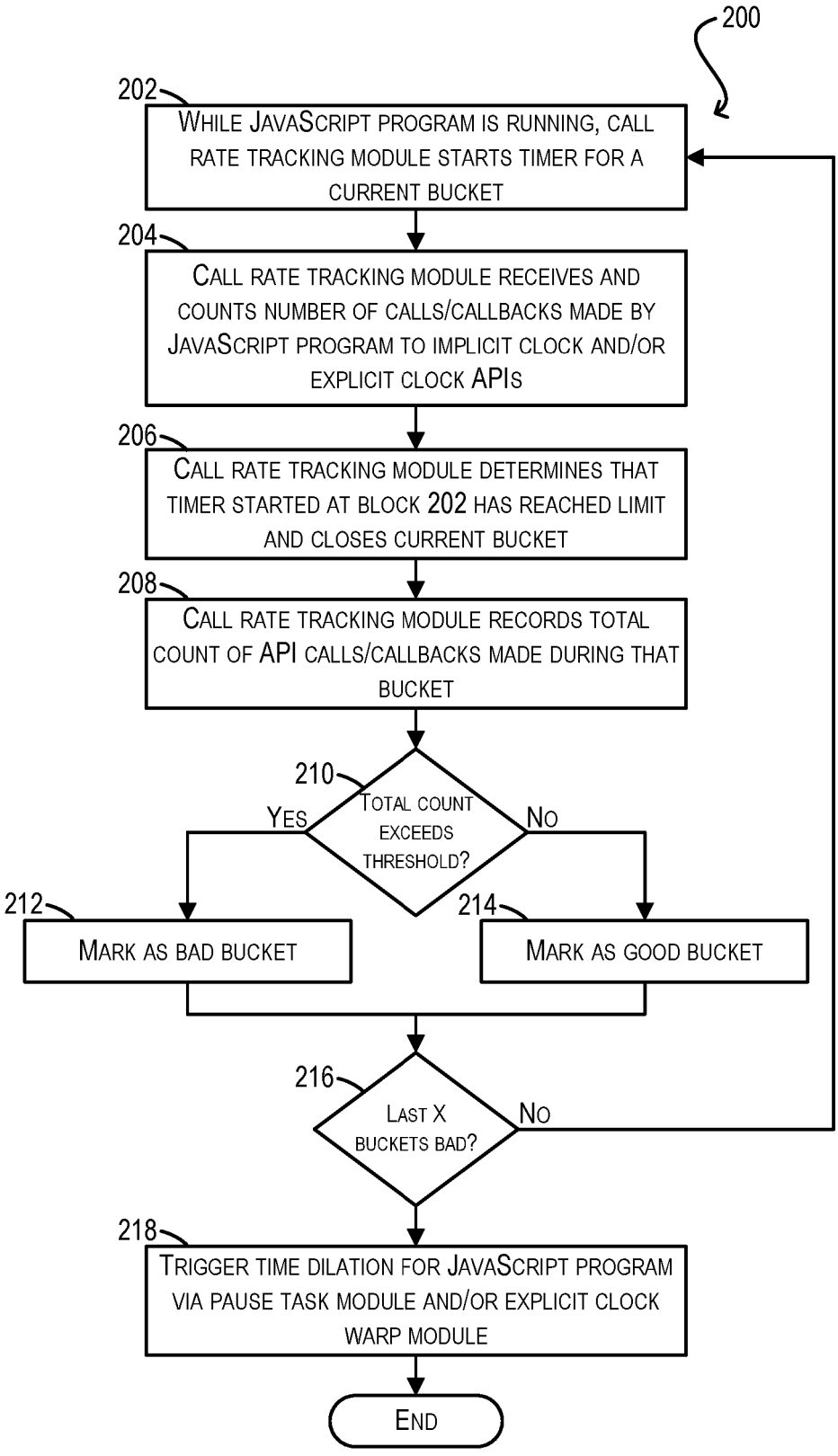
200
202
WHILE JAVASCRIPT PROGRAM IS RUNNING, CALL RATE TRACKING MODULE STARTS TIMER FOR A CURRENT BUCKET
204
CALL RATE TRACKING MODULE RECEIVES AND COUNTS NUMBER OF CALLS/CALLBACKS MADE BY JAVASCRIPT PROGRAM TO IMPLICIT CLOCK AND/OR EXPLICIT CLOCK APIS
206
CALL RATE TRACKING MODULE DETERMINES THAT TIMER STARTED AT BLOCK 202 HAS REACHED LIMIT AND CLOSES CURRENT BUCKET
208
CALL RATE TRACKING MODULE RECORDS TOTAL COUNT OF API CALLS/CALLBACKS MADE DURING THAT BUCKET
210
TOTAL COUNT EXCEEDS THRESHOLD?
YES
NO
212
MARK AS BAD BUCKET
214
MARK AS GOOD BUCKET
216
LAST X BUCKETS BAD?
NO
218
TRIGGER TIME DILATION FOR JAVASCRIPT PROGRAM VIA PAUSE TASK MODULE AND/OR EXPLICIT CLOCK WARP MODULE
END

FIG. 2

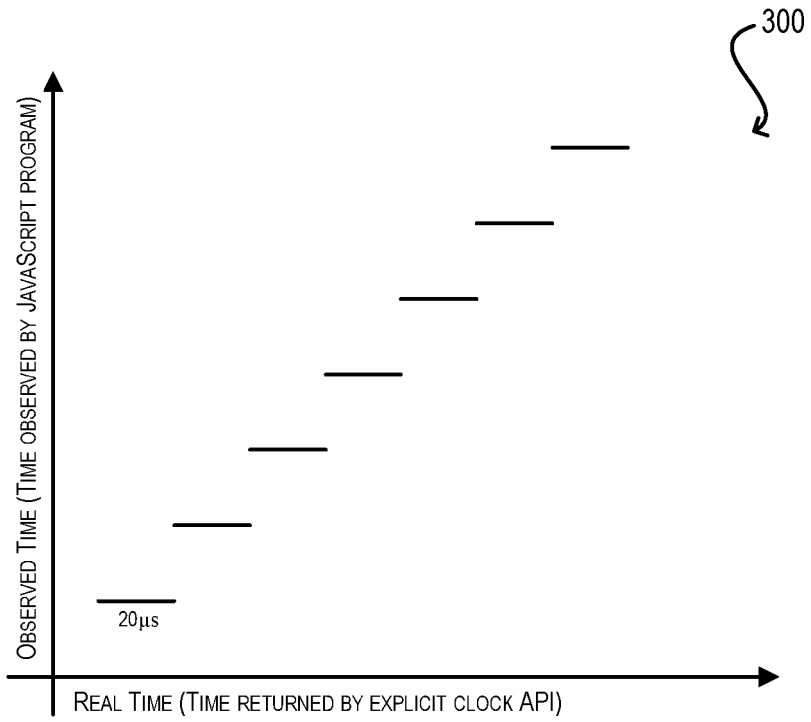
300
OBSERVED TIME (TIME OBSERVED BY JAVASCRIPT PROGRAM)
20µs
REAL TIME (TIME RETURNED BY EXPLICIT CLOCK API)

FIG. 3A

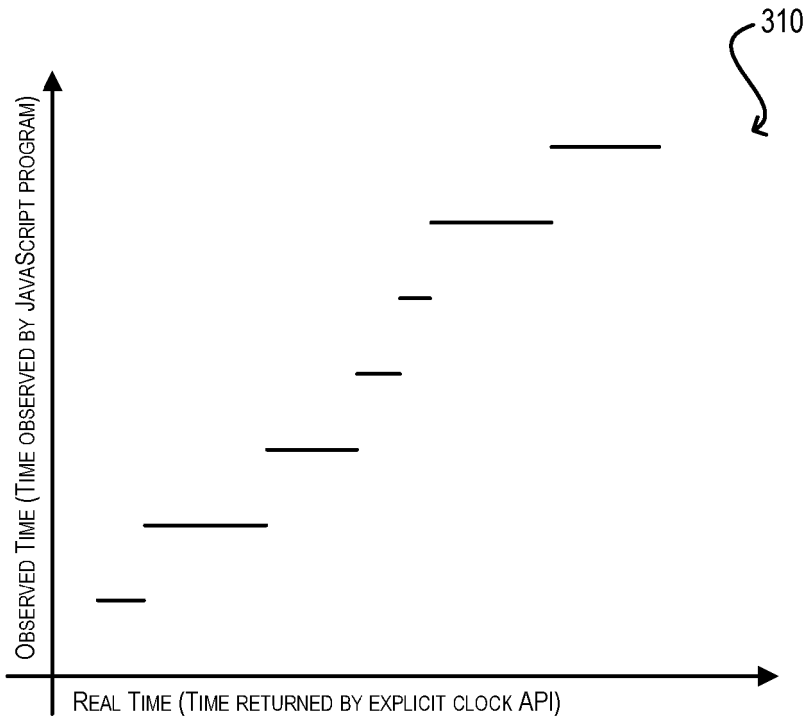
310
OBSERVED TIME (TIME OBSERVED BY JAVASCRIPT PROGRAM)
REAL TIME (TIME RETURNED BY EXPLICIT CLOCK API)

FIG. 3B

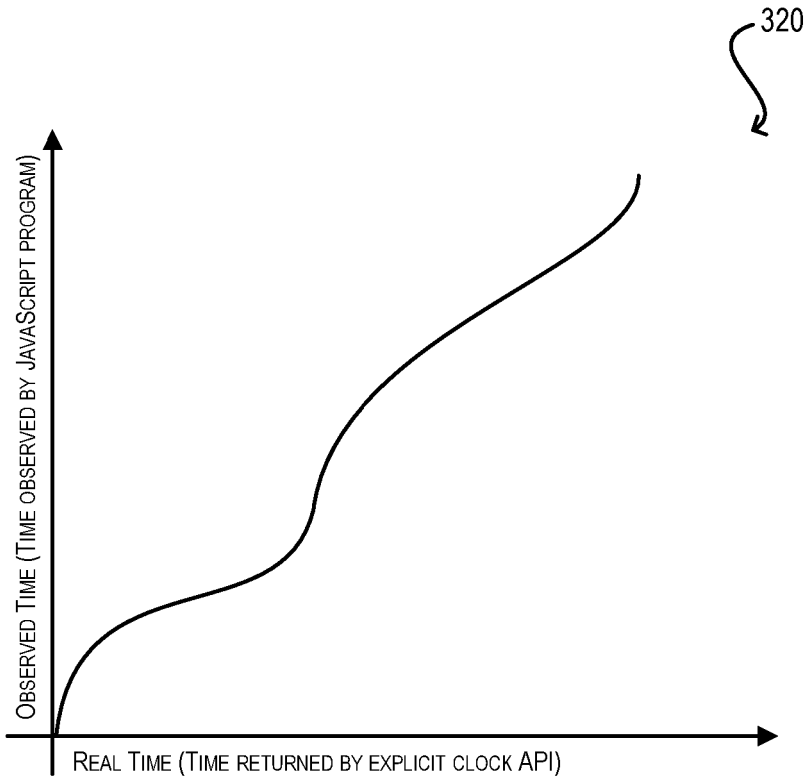
320
OBSERVED TIME (TIME OBSERVED BY JAVASCRIPT PROGRAM)
REAL TIME (TIME RETURNED BY EXPLICIT CLOCK API)

FIG. 3C

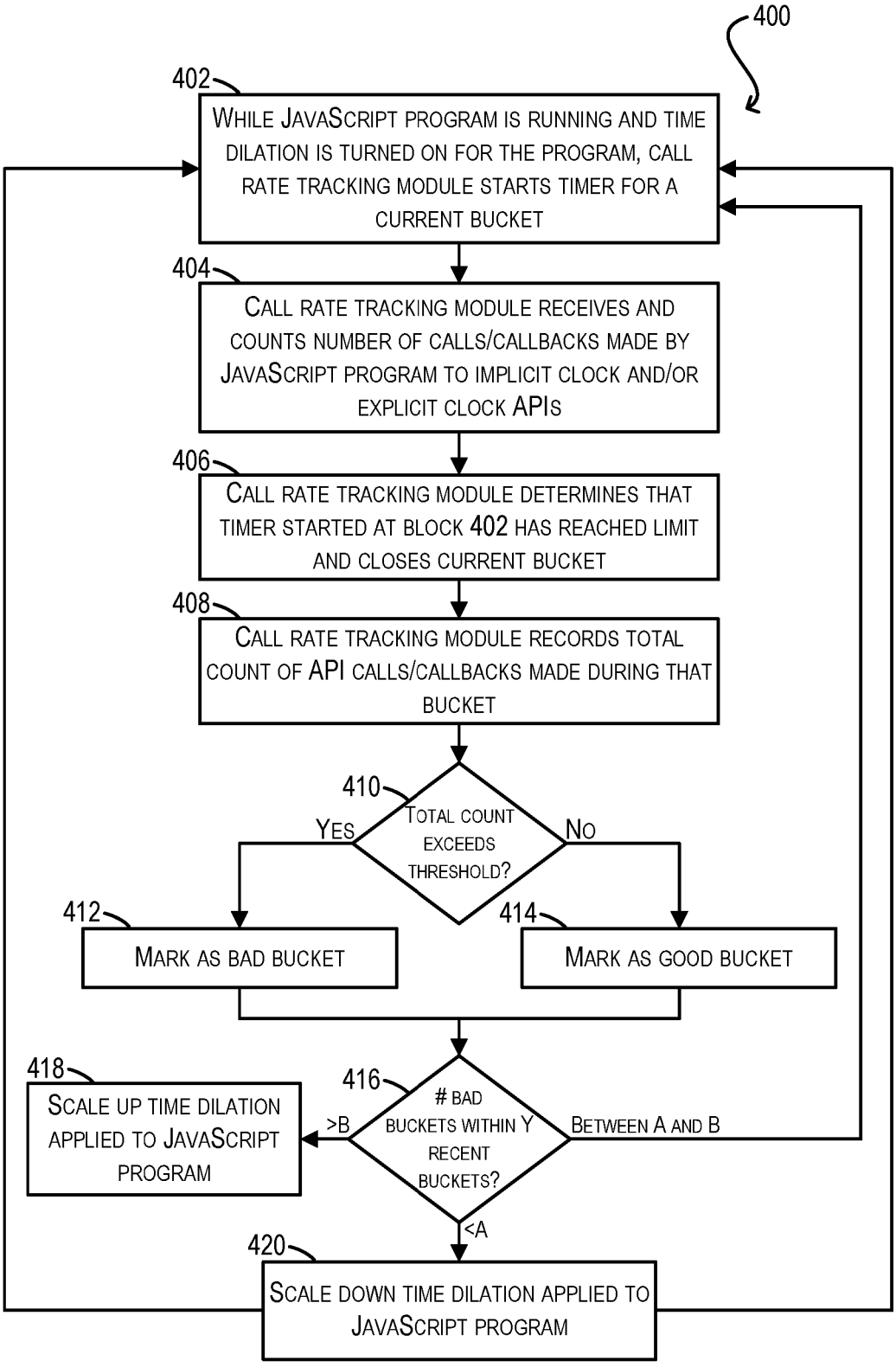
400
402
WHILE JAVASCRIPT PROGRAM IS RUNNING AND TIME DILATION IS TURNED ON FOR THE PROGRAM, CALL RATE TRACKING MODULE STARTS TIMER FOR A CURRENT BUCKET
404
CALL RATE TRACKING MODULE RECEIVES AND COUNTS NUMBER OF CALLS/CALLBACKS MADE BY JAVASCRIPT PROGRAM TO IMPLICIT CLOCK AND/OR EXPLICIT CLOCK APIS
406
CALL RATE TRACKING MODULE DETERMINES THAT TIMER STARTED AT BLOCK 402 HAS REACHED LIMIT AND CLOSES CURRENT BUCKET
408
CALL RATE TRACKING MODULE RECORDS TOTAL COUNT OF API CALLS/CALLBACKS MADE DURING THAT BUCKET
410
TOTAL COUNT EXCEEDS THRESHOLD?
YES
NO
412
MARK AS BAD BUCKET
414
MARK AS GOOD BUCKET
416
BAD BUCKETS WITHIN Y RECENT BUCKETS?
>B
BETWEEN A AND B
<A
418
SCALE UP TIME DILATION APPLIED TO JAVASCRIPT PROGRAM
420
SCALE DOWN TIME DILATION APPLIED TO JAVASCRIPT PROGRAM

FIG. 4

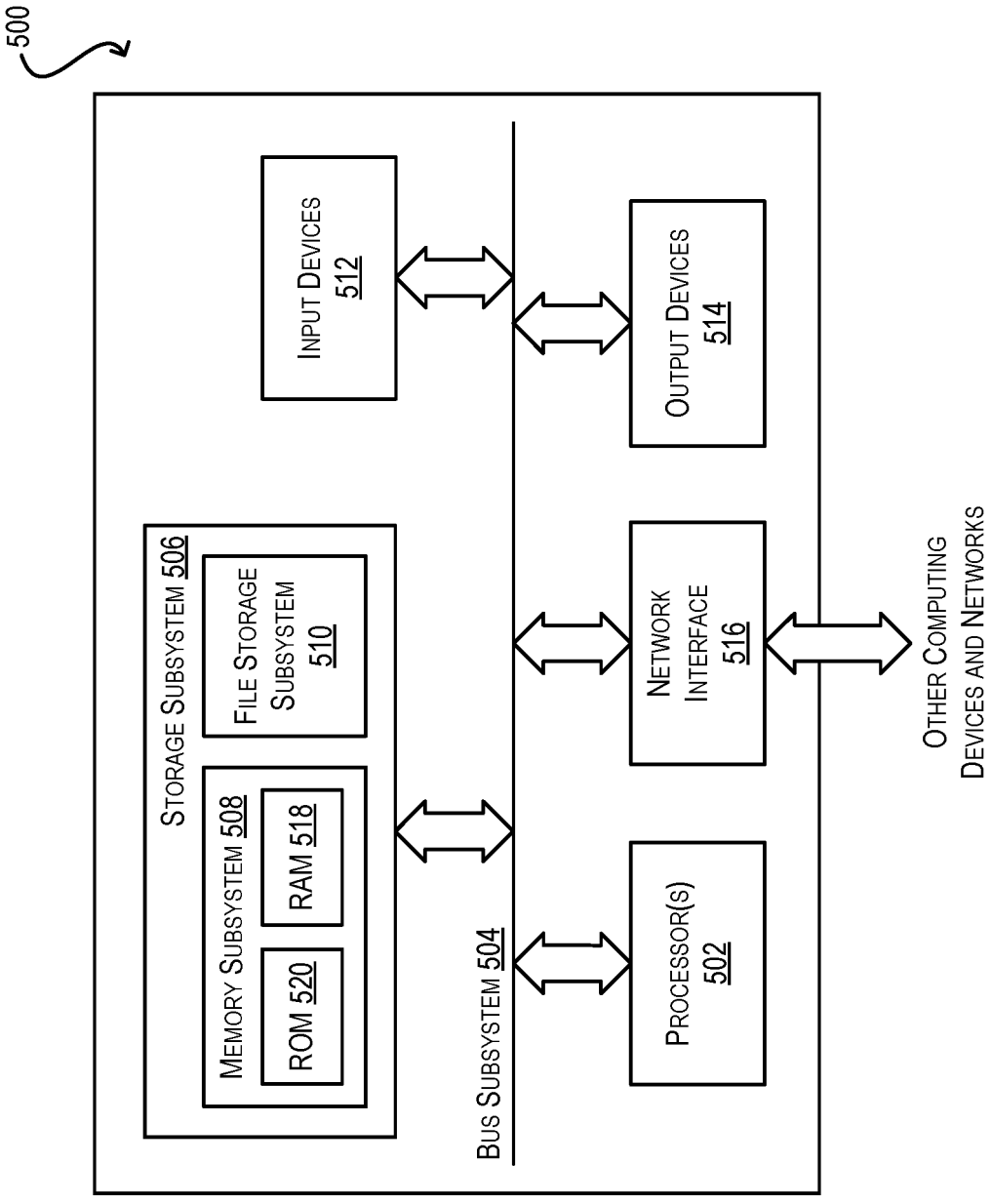
500
INPUT DEVICES 512
OUTPUT DEVICES 514
STORAGE SUBSYSTEM 506
FILE STORAGE SUBSYSTEM 510
MEMORY SUBSYSTEM 508
RAM 518
ROM 520
NETWORK INTERFACE 516
BUS SUBSYSTEM 504
PROCESSOR(S) 502
OTHER COMPUTING DEVICES AND NETWORKS

FIG. 5

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- WO 2013172913 A2 **[0003]**
- US 2014380474 A1 **[0003]**

Non-patent literature cited in the description

- Predictive black-box mitigation of timing channels. **ASLAN ASKAROV et al.** PROCEEDINGS OF THE 17TH ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY. ACM, 04 October 2010 **[0003]**