(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 January 2014 (16.01.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/008961 A1

(51) **International Patent Classification**:
G06F 21/56 (2013.01) *G06F 21/51* (2013.01)
G06F 21/62 (2013.01)

(21) **International Application Number**:
PCT/EP2012/076161

(22) **International Filing Date**:
19 December 2012 (19.12.2012)

(25) **Filing Language**: English

(26) **Publication Language**: English

(30) **Priority Data**:
61/670,000 10 July 2012 (10.07.2012) US

(71) **Applicant**: **TELEFONAKTIEBOLAGET L M ERICSSON (PUBL)** [SE/SE]; S-164 83 Stockholm (SE).

(72) **Inventors**: **LANTZ, Patrik**; Plåslagarevägen 3B, S-227 30 Lund (SE). **JOHANSSON, Björn**; Södra Villavägen 2, S-235 37 Bjärred (SE). **SMEETS, Bernard**; Dalbackavägen 11, S-247 51 Dalby (SE).

(74) **Agent**: **RÖTHINGER, Rainer**; Schweigerstrasse 2, 81541 München (DE).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

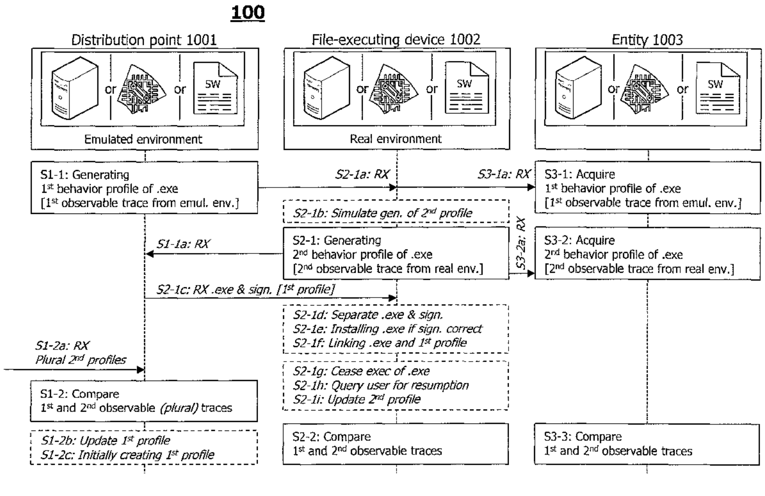
(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) Title: TECHNIQUE FOR DETERMINING A MALIGN OR NON-MALIGN BEHAVIOR OF AN EXECUTABLE FILE

Fig. 2


100
Distribution point 1001
File-executing device 1002
Entity 1003
or
or
SW
Emulated environment
Real environment
S1-1: Generating
1st behavior profile of .exe
[1st observable trace from emul. env.]
S2-1a: RX
S3-1a: RX
S3-1: Acquire
1st behavior profile of .exe
[1st observable trace from emul. env.]
S2-1b: Simulate gen. of 2nd profile
S1-1a: RX
S2-1: Generating
2nd behavior profile of .exe
[2nd observable trace from real env.]
S3-2a: RX
S3-2: Acquire
2nd behavior profile of .exe
[2nd observable trace from real env.]
S2-1c: RX .exe & sign. [1st profile]
S2-1d: Separate .exe & sign.
S2-1e: Installing .exe if sign. correct
S2-1f: Linking .exe and 1st profile
S1-2a: RX
Plural 2nd profiles
S2-1g: Cease exec of .exe
S2-1h: Query user for resumption
S2-1i: Update 2nd profile
S1-2: Compare
1st and 2nd observable (plural) traces
S1-2b: Update 1st profile
S1-2c: Initially creating 1st profile
S2-2: Compare
1st and 2nd observable traces
S3-3: Compare
1st and 2nd observable traces

(57) Abstract: A technique for determining a malign or non-malign behavior of an executable file is disclosed. In a first method aspect, the method comprises the steps of first acquiring a first behavior profile of the executable file, the first behavior profile comprising a first observable execution trace of the executable file from an emulated environment, second acquiring a second behavior profile of the executable file, the second behavior profile comprising a second observable execution trace of the executable file from a real environment, and comparing the first and second observable execution traces so as to determine the malign or non-malign behavior of the executable file. In another method aspect, the method comprises the steps of receiving a trigger condition, collecting, responsive to the trigger condition, first and second behavior profiles of the executable file from first and second one of two or more file-execution devices, the first and second behavior profiles comprising first and second observable execution traces of the executable file, and the first and second observable execution traces being non-mapped to the first and second file- execution device, respectively.

Technique for Determining a Malign or Non-Malign Behavior of an Executable File.

Technical Field

The present disclosure generally relates to determining a malign or non-malign behavior of an executable file.

Background

Since the appearance of the iPhone™ smartphone the word "app" has become synonym for applications that users of smartphone can use for various tasks. Although applications can be pre-stored on a smartphone (or similar device), the most common case is that a user downloads his/her application and installs it on his/her device. This is, for example, true for iPhone™, Android™, or Windows™ Phone devices. For instance, iPhone™ users download their applications from Apple™'s App Store, and Android™ phone users from Google™ Play although the latter users also can load their applications from other sources.

A problem with the possibility to download and install programs is the threat that the programs may be malware, that is they contain functionality that is designed to exhibit malign functions (e.g. theft of user information, corrupt user data, or code that modifies the working of other applications or even system functions) or it functions in a way that is not conforming the owner of the device (e.g. a phone used by a corporate user).

Apple™ has partially addressed this issue by introducing mandatory digital signing of applications through its application distribution point App™ Store and by screening applications before releasing them on App™ Store. For Android™ phones Google™ operates Google™ Play as a common distribution point and recently started an automated screening of applications before being made available via Google™ Play. Also Android™ applications are digitally signed.

When detecting anomalies of application or system execution, it is often desired to have or collect information about a host device. Scenarios where this is crucial involve targeted attacks where malware triggers its malicious functionality when located in a specific world-region or using a specific connectivity provider, but also for debugging purpose of crashed executables.

For the sake of simplicity in this disclosure, the term "malware" denotes all types of programs they either contain or entirely consist of functionality aiming a task that when performed successfully causes harm to the owner of the device where it operates or to the organization where the device is in operation. Well-known types of malware are virus and Trojan-horse programs but also programs like remote device managers can become malware when being installed and operated without proper authorization.

While the screening efforts of the applications by, for example, Apple™ and Google™, has a sanitizing effect on the employment of applications that a user can choose from at the official distribution points, there are still applications at the distribution points that are malware.

There are several reasons for this and most importantly one has to deal with:
- Human failures, e.g. when manually screening applications or writing the screening tool for automated screening;
- Screening covers only a set of known malign effects in the app code and thus new constructs may be undetected;
- Code obfuscation that could be used to protected applications from being exposed to software piracy can be used to hide malware functions that thus cannot be detected through code inspection;
- The malware code is devised to evade malware detection by behaving properly when being executed in a detection environment.

There exist reputation based systems for applications where people can express their opinion on an application and there are solutions where the people's approval can be secured through digital signatures. Such schemes can help to prevent a wide spread of bad applications but have the disadvantage that they are slow in detecting malware, susceptible to false opinion insertion, and that it is in practice to setup schemes that are reliable/secure. Today most distribution points have means where users can rate the app and leave any opinion in an unprotected way.

Summary

Accordingly, there is a need for an implementation of a scheme that avoids one or more of the problems discussed above, or other related problems.

In a first aspect, there is provided a method for determining a malign or non-malign behavior of an executable file, wherein the method comprises the steps of first acquiring a first behavior profile of the executable file, the first behavior profile comprising a first observable execution trace of the executable file from an emulated environment, second acquiring a second behavior profile of the executable file, the second behavior profile comprising a second observable execution trace of the executable file from a real environment and comparing the first and second observable execution traces so as to determine the malign or non-malign behavior of the executable file.

In optional refinements of the first aspect, there is or are provided at least one of the following:

- the method is performed in a file-execution device comprising the real environment, further comprising in the first acquiring receiving, from a distribution point comprising the emulated environment, both the first behavior profile and the executable file, wherein, in the second acquiring, second behavior profile is generated in the file-execution device;
- the receiving step further comprises receiving, along with the first behavior profile and the executable file, a signature of the first behavior profile;
- the signature is the result of application of a the private key of a private/public key cryptosystem, and the public key used for verifying the signature is stored on the file-execution device or is received as a part of a digital certificate;
- the method further comprises separating the executable file from the signed first behavior profile;
- the method further comprises installing, at the file-execution device, the separated executable file under the prerequisite that the separated signature is verified as correct, and linking the separated first profile to the executable file;
- the method further comprises, if the comparing step yields deviations between the first and second observable execution traces ceasing execution of the executable file, querying the user whether the ceased execution is to be resumed, and updating the second behavior profile based on the result of the query;
- the method further comprises, prior to generating the second behavior profile, simulating the result of the generating step;

- the method is performed in a distribution point comprising the real environment, wherein the executable file is pre-stored in the distribution point, further comprising in the second acquiring receiving, from a file-execution device comprising the real environment, the second behavior profile and the executable file, wherein, in the first acquiring, the first behavior profile is generated in the distribution point;
- the receiving step further comprises receiving, from each of a plurality of file-execution devices, a separate second behavior profile, and wherein the comparing step further comprises comparing the first observable execution trace with the plurality of the second observable execution traces in the second behavior profiles, wherein the method further comprises updating or initially creating the first behavior profile based on the comparison;
- the method is performed in an entity different from a distribution point and a file-execution device, wherein the first acquiring comprises receiving the first behavior profile, and the second acquiring comprises receiving the second behavior profile; and/or
- the first and second profiles are generated based on one of a treemap and a behavior graph.

In a second aspect, there is provided a method for anonymously collecting behavior data of an executable file, wherein the executable file is resident on each of two or more file-execution devices and distributed by a distribution point, and wherein the method is performed in an entity different from the distribution point and two or more file-execution devices, and comprises the steps of determining a trigger condition; first collecting, responsive to the trigger condition, a first behavior profile of the executable file from a first one of the two or more file-execution devices, the first behavior profile comprising a first observable execution trace of the executable file, and the first observable execution trace being non-mapped to the first file-execution device; and second collecting, responsive to the trigger condition, a second behavior profile of the executable file from a second one of the two or more file-execution devices, the second behavior profile comprising a second observable execution trace of the executable file, and the second observable execution trace being non-mapped to the second file-execution device.

Concerning the terminology used for the second to third, sixth and seventh aspects (as well as the other aspects insofar related to the first-named aspects), the following applies:

- The term "non-mapped" may in certain implementations be interpreted such that that the information is prevented from being mapped to, or associated with a specific user or device (e.g., in order to preserve privacy) by means of a scheme. As a non-limiting example, the scheme may comprise a (homomorphic) encryption.

- The term "executable file" may cover an app(lication) and/or a system update functionality.
- The term "behavior profile" may comprise a behavior of an executable file (in the sense of a behavioral observable trace) and/or device-specific information/settings (which may influence the behavior of the executable file as well).

In optional refinements of the second aspect, there is or are provided at least one of the following:

- the trigger condition is one of the executable file throwing a non-maskable interrupt, the two or more file-execution devices throwing a non-maskable interrupt, and the two or more file-execution devices passing a predetermined geographic location;

- the trigger condition is a comparison result determining a malign behavior of the executable file;

- the first and second collecting steps each comprise transmitting a request for anonymous collection of behavior data of the executable file to the first and second file-execution devices, and receiving the first and second observable execution traces from the first and second file-execution devices;

- the transmitting step further transmits the trigger condition, and the receiving step is a push operation from the first and second file-execution devices upon fulfillment of the trigger condition;

- the transmitting step is performed upon fulfillment of the trigger condition, and the receiving step is a pull operation initiated by the entity;

- the first and second observable execution traces being non-mapped to the first and second file-execution devices comprises the first and second observable execution traces being respectively homomorphically encrypted;

- the method further comprises buffering and obfuscating the received first and second homomorphically encrypted observable execution traces;

- the obfuscating step comprises one of adding, merging and mixing the first and second homomorphically encrypted observable execution traces;

- the method further comprises transmitting the buffered and obfuscated first and second homomorphically encrypted observable execution traces to a third party to be decrypted by a private key of a public/private key pair agreed between the file-

execution devices and the third party, the homomorphic encryption having been performed by a public key of the public/private key pair;
- the entity is a Host Information Center, HIC;
- the HIC is comprised in a remote execution device;
- the remote execution device is a cloud.

In a third aspect, there is provided a method for anonymously collecting behavior data of an executable file distributed by a distribution point, wherein the method is performed in a file-execution device, the executable file is resident on the file-execution device, and comprises the steps of receiving, from an entity different from the distribution point and the file-execution device, a request for anonymous collection of behavior data of the executable file, and collecting, responsive to the received request, a behavior profile of the executable file, the behavior profile comprising an observable execution trace of the executable file, and the observable execution trace being non-mapped to the file-execution device.

In optional refinements of the third aspect, there is or are provided at least one of the following:
- the method further comprises homomorphically encrypting the collected observable execution trace so as to be non-mapped to the file-execution device;
- the encrypting step utilizes a public key of a public/private key pair agreed between the file-execution device and a third party;
- the method further comprises transmitting the collected observable execution trace to the entity;
- the receiving step further receives a trigger condition, and the transmitting step is a push operation from the file-execution device upon fulfillment of the trigger condition;
- the transmitting step is a pull operation initiated by the entity;
- the file-execution device is comprised in a trusted environment, and the executable file is a trusted application;
- the file-execution device performs the collecting step under supervision of a hypervizor entity.

In a fourth aspect, there is provided a computer program product comprising program code portions for performing a method according to any one of the first to third aspects, when the computer program product is executed on one or more computing devices.

In an optional refinement of the fourth aspect, the computer program product is stored on a computer readable recording medium.

In a fifth aspect, there is provided an apparatus for determining a malign or non-malign behavior of an executable file, the apparatus comprising at least one processor configured to acquire a first behavior profile of the executable file, the first behavior profile comprising a first observable execution trace of the executable file from an emulated environment, acquire a second behavior profile of the executable file, the second behavior profile comprising a second observable execution trace of the executable file from a real environment, and compare the first and second observable execution traces so as to determine the malign or non-malign behavior of the executable file.

In a sixth aspect, there is provided an apparatus for anonymously collecting behavior data of an executable file, wherein the apparatus is constituted by an entity different from a distribution point and two or more file-execution devices, the executable file is resident on each of the two or more file-execution devices, and the apparatus comprises at least one processor configured to determine a trigger condition, collect, responsive to the trigger condition, a first behavior profile of the executable file from a first one of the two or more file-execution devices, the first behavior profile comprising a first observable execution trace of the executable file, and the first observable execution trace being non-mapped to the first file-execution device, and collect, responsive to the trigger condition, a second behavior profile of the executable file from a second one of the two or more file-execution devices, the second behavior profile comprising a second observable execution trace of the executable file, and the second observable execution trace being non-mapped to the second file-execution device.

In a seventh aspect, there is provided an apparatus for anonymously collecting behavior data of an executable file, wherein the apparatus is constituted by a file-execution device, the executable file is resident on the file-execution device, and the apparatus comprises at least one processor configured to receive, from an entity different from a distribution point and the file-execution device, a request for anonymous collection of behavior data of the executable file, and collect, responsive to the received request, a behavior profile of the executable file, the behavior profile comprising an observable execution trace of the executable file, and the observable execution trace being non-mapped to the file-execution device.

In an eighth aspect, there is provided a system, comprising the apparatus according to the fifth aspect being functionally split between a distribution point and a file-execution device, wherein the comparing operation is performed in at least one of the distribution point and the file-execution device, and a secure channel is established between the distribution point and the file-execution device.

In a ninth aspect, there is provided a data structure for storing observable execution traces of an executable file in a behavior profile, the data structure comprising at least one entry per system call performed by the executable file, the entry comprising an argument-to-function/method call and a timestamp of when the system call occurred.

In an optional refinement of the ninth aspect, there is or are provided at least one of the following:
- the system call comprised in the entry is hashed, excluding the timestamp;
- the argument is constituted by a classification of the argument; and/or
- the classification is a list of data elements comprising at least one of an origin of the argument, a security label, and least one argument range constraint.

Brief Description of the Drawings

The embodiments of the technique presented herein are described herein below with reference to the accompanying drawings, in which:

Fig. 1 shows components comprised in a first exemplary device embodiment realized in the form of a distribution point, a file-executing device or another entity;

Fig. 2 shows a method embodiment which also reflects the interaction between the components of the device embodiment;

Fig. 3 shows a data structure embodiment;

Fig. 4A shows a first exemplary implementation of the embodiment in the form of a treemap;

Fig. 4B shows a second exemplary implementation of the embodiment in the form of a behavior graph; and

Fig. 5 shows components and method steps comprised in a second exemplary device and method embodiment realized in the form of a distribution point or a file-executing device.

Detailed Description

In the following description, for purposes of explanation and not limitation, specific details are set forth (such as particular signaling steps) in order to provide a thorough understanding of the technique presented herein. It will be apparent to one skilled in the art that the present technique may be practiced in other embodiments that depart from these specific details. For example, the embodiments will primarily be described in the context of so-called "apps" as an example for executable files; however, this does not rule out the use of the present technique in connection with other file systems or formats.

For the purpose of this disclosure, the terms "apparatus" and "system" have been introduced. Without being restricted thereto, the "system" may be implemented as a wireless communication network or a portion thereof. Moreover, the "apparatus" or the "wireless communication device" may be functionally split into a "distribution point" and a "file-execution device". In turn, the "distribution point" may be implemented as functionality in the Internet, for example in the IT/Telecommunications cloud. Moreover, the "file-execution device" may be fixed/wirebound or mobile, such as a fixed workstation, or a fixed or wireless desktop/ laptop, or a fixed or mobile Machine-to-Machine (M2M) interface, or a mobile terminal, such as a smartphone. However, those implementation examples are only illustrative; the person skilled in the art can readily devise various additional or supplemental implementations of the "system" and "wireless communication device".

Moreover, those skilled in the art will appreciate that the services, functions and steps explained herein may be implemented using software functioning in conjunction with a programmed microprocessor, or using an Application Specific Integrated Circuit (ASIC), a Digital Signal Processor (DSP) or general purpose computer. It will also be appreciated that while the following embodiments are described in the context of methods and devices, the technique presented herein may also be embodied in a computer program product as well as in a system comprising a computer processor and a memory coupled to the processor, wherein the memory is encoded with one or more programs that execute the services, functions and steps disclosed herein.

The present disclosure, without being restricted thereto, may be summarized in that the fact is used that the app is screened, preferably dynamically by executing it, and that there is a known trusted distribution point that could convey its findings of the screening in a more relevant way. Today the fact an app is made downloadable via a distribution point implies that the screening did not find anything harmful in the code having testing it. In the case of Android™, one also lists the permission (to other functions) the app requires. But this is basically all information that is available. According to the present disclosure, the user can of the device be actually instructed of the expected (and approved observed) behavior of the application. In the device, the app can be monitored when it executes and compare it with the behavior it showed during the screening. This allows to identify security relevant deviations from the approved behavior and to take countermeasures, e.g. blocking the app from further execution, notifying the user and or distribution point. Through digital signing the distribution point can convey the observed behavior in a secure (integrity protected) way. When identifying deviating behavior, device-specific information may be collected from the device in order to determine parameters that may have caused the abnormal behavior.

Fig. 1 shows components comprised in a first exemplary device embodiment realized in the form of a distribution point 1001, a file-executing device 1002 or another entity 1003. As shown in Fig. 1, the distribution point 1001 comprises a core functionality (e.g., one or more of a Central Processing Unit (CPU), dedicated circuitry and/or a software module) 10011, an optional memory (and/or database) 10012, an optional transmitter 10013 and an optional receiver 10014. Moreover, the distribution point 1001 comprises an acquirer 10015, a comparator 10016, an optional updater 10017 and an optional creator 10018.

Likewise, the file-executing device 1002 comprises a core functionality (e.g., one or more of a Central Processing Unit (CPU), dedicated circuitry and/or a software module) 10021, an optional memory (and/or database) 10022, an optional transmitter 10023 and an optional receiver 10024. Moreover, the device 1002 comprises an acquirer 10025, a comparator 10026, an optional separator 10027, an optional installer 10028, an optional linker 10029, an optional executioner 100210, an optional query 200211, an optional updater 100212 and an optional simulator 100213.

Finally, the (other) entity 1003 comprises a core functionality (e.g., one or more of a Central Processing Unit (CPU), dedicated circuitry and/or a software module) 10031, an optional memory (and/or database) 10032, an optional transmitter 10033 and an optional receiver 10034. Moreover, the entity 1003 comprises an acquirer 10035 and a comparator 10036.

In the following paragraphs, assume that x = 1, 2 or 3. As partly indicated by the dashed extensions of the functional block of the CPUs 100x1, the acquirer 10015, the comparator 10016, the updater 10017 and the creator 10018 (of the Distribution point 1001), the acquirer 10025, the comparator 10026, the separator 10027, the installer 10028, the linker 10029, the executioner 100210, the query 200211, the updater 100212 and the simulator 100213 (of the device 1002) and the acquirer 10035 and the comparator 10036 (of the entity 1003) as well as the memory 100x2, the transmitter 100x3 and the receiver 100x4 may at least partially be functionalities running on the CPUs 100x2, or may alternatively be separate functional entities or means controlled by the CPUs 100x1 and supplying the same with information. The transmitter and receiver components 100x3, 100x4 may be realized to comprise suitable interfaces and/or suitable signal generation and evaluation functions.

The CPUs 100x1 may be configured, for example, using software residing in the memories 100x2, to process various data inputs and to control the functions of the memories 100x2, the transmitter 100x3 and the receiver 100x3 (as well the acquirer 10015, the comparator 10016, the updater 10017 and the creator 10018 (of the Distribution point 1001), the acquirer 10025, the comparator 10026, the separator 10027, the installer 10028, the linker 10029, the executioner 100210, the query 200211, the updater 100212 and the simulator 100213 (of the device 1002) and the acquirer 10035 and the comparator 10036 (of the entity 1003)). The memory 100x2 may serve for storing program code for carrying out the methods according to the aspects disclosed herein, when executed by the CPUs 100x1.

It is to be noted that the transmitter 100x3 and the receiver 100x4 may be provided as an integral transceiver, as is indicated in Fig. 1. It is further to be noted that the transmitters/receivers 10013, 10014 may be implemented as physical transmitters/receivers for transceiving via an air interface or a wired connection, as routing/forwarding entities/interfaces between network elements, as functionalities for writing/reading information into/from a given memory area or as any suitable combination of the above. At least one of the above-described the acquirer 10015, the comparator 10016, the updater 10017 and the creator 10018 (of the Distribution point 1001), the acquirer 10025, the comparator 10026, the separator 10027, the installer 10028, the linker 10029, the executioner 100210, the query 200211, the updater 100212 and the simulator 100213 (of the device 1002) and the acquirer 10035 and the comparator 10036 (of the entity 1003), or the respective functionalities, may also be implemented as a chipset, module or subassembly.

Fig. 2 illustrates an embodiment of a method for managing connection states of at least two subscriptions. In the signaling diagram of Fig. 2, time aspects between signaling are reflected in the vertical arrangement of the signaling sequence as well as in the sequence numbers. It is to be noted that the time aspects indicated in Fig. 2 do not necessarily restrict any one of the method steps shown to the step sequence outlined in Fig. 2. This applies in particular to method steps that are functionally disjunctive with each other.

The embodiment may be based on a secure channel between the mobile 1002 and a Trusted Service 1001 in the network. Part of the embodiment may reside in establishing a security context between the mobile 1002 and the trusted network service 1001. As a best mode, there is disclosed a setup where the trusted service 1001 (or the distribution point) signs the profile data with the secret key of a public-key cryptosystem. The public key that can be used for verifying the signature may be either already stored on the device 1002 or may be sent as part of a so-called (digital) certificate whose content can be verified by chain of certificates in a PKI (Public Key Infrastructure) scheme whose root certificate is stored on the device 1001.

Fig. 3 shows a Data Structure (DS) embodiment, which may be stored in at least one of the memories 10012, 10022, 10032.

One way to analyze an app(lication) is to perform a static code analysis. Such an analysis can detect leaks of private information. However, static analysis is limited as it may not cover the dynamics of the application as it executes or it is rendered ineffective due to hiding and code obfuscation techniques.

The compilation of the app behavior profile P1, P2 is conducted by a behavior analysis in the distribution point before releasing it to the public. The app is executed in an emulated environment. During its execution, any interactions with the underlying operating system by the app are observed and stored in the profile P1, P2 using e.g. taint analysis. But other methods to capture the behavior are also possible, as long as they deliver a digitally observable execution trace of the behavior. The data of this profile P1 may be referred to as the Reference Application Behavior Profile (RABP).

This profile P1 may later be verified against a profile P2 generated by the same app on a real mobile device 1002. To augment the effectiveness of the profiles P1, P2 one can even watch the argument to function/method calls, e.g.

Call_profile_entry E = timestamp + syscall nr + argument 1 +... + argument n
or
Call_profile_entry E = hash (timestamp + syscall nr + argument 1 +... + argument n)

for each system call that the app generates. Instead of the argument itself, it would also be possible to first perform a classification of the argument and then optionally include the classification of the argument in the hash computation. Such a classification could be a list of data elements like, e.g., argument origin, security label, or argument range constraints.

```
Call_profile_entry E = timestamp + syscall nr + \
                 Classify(argument 1) +... + Classify(argument n))
or
Call_profile_entry E = hash (timestamp + syscall nr + \
                 Classify(argument 1) +... + Classify(argument n)))
```

The classification may render the call-profile-entries more suitable in capturing generic arguments rather than specific values.

Fig. 4A shows a first exemplary implementation of the embodiment in the form of a treemap, and Fig. 4B shows a second exemplary implementation of the embodiment in the form of a behavior graph.

The app behavior profile itself may be composed of information collected during a behavioral analysis of the app during runtime. To this end, different technologies can be used as example embodiments we mention here treemaps in Fig. 4A and behavior graphs shown in Fig. 4B.

One further feature of these tree maps and behavior graphs resides in rendering the same e.g. multi-dimensional, so they can capture the behavior in a richer way.

Returning to Figs. 1 and 2, when a user downloads an app from the distribution point 1001, the file that contains the app code could be equipped with the signed RABP P1. Alternatively, the user could download such behavior profile from another place, e.g. a trusted service 1001 that makes behavior profiles P1 of applications. For simplicity, it can be assumed that the behavior profile P1 is bundled with the app code itself.

In the device 1002, the app file is dissected (S2-1d, 10027) in the normal app part and the signed RABP. The former is processed by the existing procedures for installing (S2-1e, 10028) the app with the additional restriction that the signature of the profile data is successfully verified as being correct. The latter, RABP P1 may be in the device 1002 and linked (S2-1f, 10029) to the app so that when the app is executed its behavior profile P1 can be found in the device.

When the app executes, the device 1002 may also trace the app as it proceeds and constructs an Observed Application Behavior Profile (OABP) P2. The OABP P2 may be compared (S1-2, S2-2, S3-3; 10016, 10026, 10036) to the RABP P1 and if the comparison reveals significant deviations the app may be stopped or halted and the user is informed and asked for consent to proceed.

Note that this allows for a mechanism where the RABP P1 may be updated by the gained insight through the user consent so the user is not bothered the next time when the same condition occurs at a later instant.

In an alternative mode, the device 1002 could first simulate (S2-1b, 100213) the upcoming behavior, i.e., opening of external connections, and first compare the resulting behavior profile P2 with the reference P1 before committing to actually actions. This avoids that improper behavior only can be detected when it already occurred.

Instead of the only analysis by the distribution point 1001, the distribution point could use a set of trusted devices 1002 that already downloaded and installed the app to improve the correctness of its behavior profile P1. These devices can report (S1-2a, 10014) their results (updated RABPs) P1 and the distribution point 1001 can compare the reports and compile a behavior profile (S1-2c, 10018) or augments a basic profile (S1-2b, 10017) that it already established from a basic screening of the app. In such a way one has a collective learning that improves the quality of the protection the reference profile provides.

The collected information must be securely transmitted from the trusted devices to the distribution point. One solution for that may be SSL/TLS or VPN secured connections.

Fig. 5 shows components and method steps comprised in a second exemplary device and method embodiment realized in the form of a distribution point or a file-executing device 1002. It is noted that the file-execution devices 1002 #1, 1002 #2 and the entity 1003 may basically have the same structure as depicted in Fig. 1. That is, e.g., a monitor 100214 comprised in the file-execution device(s) 1002 may also be a function or a separate chip/subassembly implemented in or controlled by the CPU 10021 of each file-execution device 1002. Moreover, all steps S1 to S7 may involve corresponding means implemented in or controlled by the respective CPUs; as a non-exclusive example, the obfuscating/mixing performed by the entity 1003 may involve an obfuscator/mixer (not shown).

When an anomaly has been detected during application or system execution (S3, 100214), the execution host (file-execution device) 1002 is pulled for device information (S2, S4, S4a). In another use-scenario where the execution profiles are compared locally, the host device pushes (S4, S4a) this information to the entity 1003 responsible for collecting it (abbreviated, e.g., as Host Information Collector, HIC).

The HIC 1003 can be deployed as a service in the cloud. Collected information can be merged (S5) by increasing a counter for each specific parameter present in the device-information. This information may have been sent to the HIC 1003 encrypted (S4, S4a) and may use a homomorphic encryption or another scheme that prevents the information from being mapped to (e.g., from being usable to identify) a specific user or device 1002 #1, 1002 #2 in order to preserve privacy. Sending information from a trusted application residing in a trusted execution environment would prevent tampering of device-information. If the goal is to monitor a system, a hypervizor solution can be responsible for sending (S4, S4a) the information.

The behavior above can be generalized so that trigger conditions are defined by application developers (third party) 1004 (S1). In the case when trigger conditions are met, certain device-information is pushed encrypted (S4, S4a) to the HIC 1003.

The HIC 1003 buffers and/or mixes (obfuscates, S5) the data in a way that prevents the developer from mapping received data with a certain user or device 1002 when retrieving (S6) the statistical data from the HIC 1003. In order for the HIC 1003 to merge encrypted data (S5), a homomorphic encryption scheme or other similar schemes can be applied.

More specifically, when a trigger condition is met (S3), possibly based on hypervisor monitoring, a trusted application encrypts the application developer requested device-specific data with a public key (Pub key) supplied by the developer. This encrypted information is then sent to the HIC 1003 (S1) where the third party 1004 can retrieve (S6) the merged data and decrypt it with its private key (S7). This behavior prevents the HIC 1003 from reading sensitive data and mapping users 1002 with the read data, and the third party 1004 is only able to retrieve merged data (S6) and therefore unable to map individual users.

As a non-liming example, the third party 1004 might want to collect location-information from all devices using its app when a certain condition is fulfilled, for example, to retrieve information about where customers live. One solution would be to request permission to retrieve location updates. However, this does not prevent the third party 1004 from mapping individual app users with location data which may be a privacy concern for the user. Another solution to this particular example is to ask the connectivity provider for location-data but this suggested approach is more flexible in terms of monitoring host-device execution and collecting device-information.

The present disclosure provides one or more of the following advantages:

- Alleviating the threat of malware that hides itself from being detected; the quality of applications that the user obtains is improved.
- No or minor changes in the existing way of distribution applications and their so-called eco system.
- Providing a way to identify parameters on a host device that triggers malicious functionality or causes other abnormal behavior.
- Gradually improving the quality of the protection by collective learning.
- Collecting anonymized device-information provides a way to share this information in a flexible way by applying it on different use-scenarios while preserving privacy.

It is believed that the advantages of the technique presented herein will be fully understood from the foregoing description, and it will be apparent that various changes may be made in the form, constructions and arrangement of the exemplary aspects thereof without departing from the scope of the invention or without sacrificing all of its advantageous effects. Because the technique presented herein can be varied in many ways, it will be recognized that the invention should be limited only by the scope of the claims that follow.

Claims

1. A method for determining a malign or non-malign behavior of an executable file, wherein the method comprises the steps of:

- first acquiring (S1-1, S2-1a, S3-1) a first behavior profile (P1) of the executable file, the first behavior profile comprising a first observable execution trace of the executable file from an emulated environment;

- second acquiring (S1-1a, S2-1, S3-2) a second behavior profile (P2) of the executable file, the second behavior profile comprising a second observable execution trace of the executable file from a real environment; and

- comparing (S1-2, S2-2, S3-3) the first and second observable execution traces so as to determine the malign or non-malign behavior of the executable file.

2. The method according to claim 1, wherein the method is performed in a file-execution device (1002) comprising the real environment, further comprising in the first acquiring:

- receiving (S2-1a), from a distribution point (1001) comprising the emulated environment, both the first behavior profile and the executable file,

wherein, in the second acquiring, second behavior profile is generated (S2-1) in the file-execution device.

3. The method according to claim 2, wherein the receiving step further comprises:

- receiving (S2-1c), along with the first behavior profile and the executable file, a signature of the first behavior profile.

4. The method of claim 3, wherein:

- the signature is the result of application of a the private key of a private/public key cryptosystem, and

- the public key used for verifying the signature is stored on the file-execution device or is received as a part of a digital certificate.

5. The method according to claim 3 or 4, further comprising:

- separating (S2-1d) the executable file from the signed first behavior profile.

6. The method according to claim 5, further comprising:

- installing (S2-1e), at the file-execution device, the separated executable file under the prerequisite that the separated signature is verified as correct;

- linking (S2-1f) the separated first profile to the executable file.

7. The method according to claim 5 or 6, further comprising, if the comparing step yields deviations between the first and second observable execution traces:

- ceasing execution (S2-1g) of the executable file,

- querying (S2-1h) the user whether the ceased execution is to be resumed, and

- updating (S2-1i) the second behavior profile based on the result of the query.

8. The method according to claim 2, further comprising, prior to generating the second behavior profile:

- simulating (S2-1b) the result of the generating step.

9. The method according to claim 1, wherein the method is performed in a distribution point (1001) comprising the real environment, wherein the executable file is pre-stored in the distribution point, further comprising in the second acquiring:

- receiving (S1-1a), from a file-execution device (1002) comprising the real environment, the second behavior profile and the executable file,

wherein, in the first acquiring, the first behavior profile is generated (S1-1) in the distribution point.

10. The method according to claim 9, wherein the receiving step further comprises:

- receiving (S1-2a), from each of a plurality of file-execution devices, a separate second behavior profile, and

wherein the comparing step further comprises:

- comparing (S1-2) the first observable execution trace with the plurality of the second observable execution traces in the second behavior profiles,

wherein the method further comprises:

- updating (S1-2b) or initially creating (S1-2c) the first behavior profile based on the comparison.

11. The method according to claim 1, wherein the method is performed in an entity (1003) different from a distribution point (1001) and a file-execution device (1002), wherein:

- the first acquiring comprises receiving (S3-1a) the first behavior profile, and

- the second acquiring comprises receiving (S3-2a) the second behavior profile.

12. The method according to claim 1, wherein the first and second profiles are generated based on one of a treemap and a behavior graph.

13. A method for anonymously collecting behavior data of an executable file, wherein the executable file is resident on each of two or more file-execution devices and distributed by a distribution point (1001), and wherein the method is performed in an entity (1003) different from the distribution point (1001) and two or more file-execution devices (1002 #1, 1002 #2), and comprises the steps of:

- determining (S1) a trigger condition;
- first collecting (S2, S4), responsive to the trigger condition, a first behavior profile (C1) of the executable file from a first one of the two or more file-execution devices, the first behavior profile comprising a first observable execution trace of the executable file, and the first observable execution trace being non-mapped to the first file-execution device; and
- second collecting (S2, S4a), responsive to the trigger condition, a second behavior profile (C2) of the executable file from a second one of the two or more file-execution devices, the second behavior profile comprising a second observable execution trace of the executable file, and the second observable execution trace being non-mapped to the second file-execution device.

14. The method of claim 13, wherein the trigger condition is one of:

the executable file throwing a non-maskable interrupt,

the two or more file-execution devices throwing a non-maskable interrupt, and

the two or more file-execution devices passing a predetermined geographic location.

15. The method of claims 13 or 14 and any one of claims 1 to 12, wherein the trigger condition is a comparison result determining a malign behavior of the executable file.

16. The method of any one of claims 13 to 15, wherein the first and second collecting steps each comprise:

transmitting (S2) a request for anonymous collection of behavior data of the executable file to the first and second file-execution devices; and

receiving (S4, S4a) the first and second observable execution traces from the first and second file-execution devices.

17. The method according to claim 16, wherein:

the transmitting step further transmits the trigger condition, and the receiving step is a push operation from the first and second file-execution devices upon fulfillment of the trigger condition.

18. The method according to claim 16, wherein:

the transmitting step is performed upon fulfillment of the trigger condition, and

the receiving step is a pull operation initiated by the entity.

19. The method according to any one of claims 13 to 18, wherein the first and second observable execution traces being non-mapped to the first and second file-execution devices comprises the first and second observable execution traces being respectively homomorphically encrypted.

20. The method according to claim 19, further comprising:

buffering and obfuscating (S5) the received first and second homomorphically encrypted observable execution traces.

21. The method according to claim 20, wherein the obfuscating step comprises one of adding, merging and mixing the first and second homomorphically encrypted observable execution traces.

22. The method according to claim 20 or 21, further comprising:

transmitting (S6) the buffered and obfuscated first and second homomorphically encrypted observable execution traces to a third party (1004) to be decrypted by a private key of a public/private key pair agreed between the file-execution devices and the third party, the homomorphic encryption having been performed by a public key of the public/private key pair.

23. The method according to any one of claims 13 to 22, wherein:

the entity is a Host Information Center, HIC.

24. The method according to claim 23, wherein:

the HIC is comprised in a remote execution device, such as a cloud.

25. A method for anonymously collecting behavior data of an executable file distributed by a distribution point (1001), wherein the method is performed in a file-execution device (1002 #1, 1002 #2), the executable file is resident on the file-execution device, and comprises the steps of:

- receiving (S2), from an entity (1003) different from the distribution point and the file-execution device, a request for anonymous collection of behavior data of the executable file; and

- collecting (S3), responsive to the received request, a behavior profile (C1, C2) of the executable file, the behavior profile comprising an observable execution trace of the executable file, and the observable execution trace being non-mapped to the file-execution device.

26. The method of claim 25, further comprising:

homomorphically encrypting (S3) the collected observable execution trace so as to be non-mapped to the file-execution device.

27. The method of claim 26, wherein:

the encrypting step utilizes a public key of a public/private key pair agreed between the file-execution device and a third party (1004).

28. The method of any one of claims 25 to 27, further comprising:

transmitting (S4, S4a) the collected observable execution trace to the entity.

29. The method according to claim 28, wherein:

the receiving step further receives a trigger condition, and

the transmitting step is a push operation from the file-execution device upon fulfillment of the trigger condition.

30. The method according to claim 28, wherein:

the transmitting step is a pull operation initiated by the entity.

31. The method according to any one of claims 25 to 30, wherein the file-execution device is comprised in a trusted environment, and the executable file is a trusted application.

32. The method according to any one of claims 25 to 30, wherein the file-execution device performs the collecting step under supervision of a hypervizor entity.

33. The method according to any one of claims 13 to 32, wherein each behavior profile further comprises device information on the respective file-execution device.

34. A computer program product comprising program code portions for performing a method according to any one of the preceding claims, when the computer program product is executed on one or more computing devices.

35. The computer program product of claim 34, stored on a computer readable recording medium.

36. A wireless communication device (1001, 1002, 1003) for determining a malign or non-malign behavior of an executable file, the wireless communication device comprising at least one processor (10011, 10021, 10031) configured to:

- acquire a first behavior profile (P1) of the executable file, the first behavior profile comprising a first observable execution trace of the executable file from an emulated environment;
- acquire a second behavior profile (P2) of the executable file, the second behavior profile comprising a second observable execution trace of the executable file from a real environment; and
- compare the first and second observable execution traces so as to determine the malign or non-malign behavior of the executable file.

37. A wireless communication device for anonymously collecting behavior data of an executable file, wherein the apparatus is constituted by an entity (1003) different from a distribution point (1001) and two or more file-execution devices (1002 #1, 1002 #2), the executable file is resident on each of the two or more file-execution devices, and the wireless communication device comprises at least one processor (10031) configured to:

- determine a trigger condition,
- collect, responsive to the trigger condition, a first behavior profile (C1) of the executable file from a first one of the two or more file-execution devices, the first behavior profile comprising a first observable execution trace of the executable file, and the first observable execution trace being non-mapped to the first file-execution device, and
- collect (S2, S4a), responsive to the trigger condition, a second behavior profile (C2) of the executable file from a second one of the two or more file-execution devices, the second behavior profile comprising a second observable execution trace of the executable file, and the second observable execution trace being non-mapped to the second file-execution device.

38. A wireless communication device for anonymously collecting behavior data of an executable file, wherein the wireless communication device is constituted by a file-execution device (1002 #1, 1002 #2), the executable file is resident on the file-execution device, and the wireless communication device comprises at least one processor (10021) configured to:

- receive, from an entity (1003) different from a distribution point (1001) and the file-execution device, a request for anonymous collection of behavior data of the executable file; and
- collect, responsive to the received request, a behavior profile (C1, C2) of the executable file, the behavior profile comprising an observable execution trace of the executable file, and the observable execution trace being non-mapped to the file-execution device.

39. A system (100), comprising:

- the wireless communication device according to claim 36 being functionally split between a distribution point and a file-execution device,

wherein:

the comparing operation is performed in at least one of the distribution point and the file-execution device, and

a secure channel is established between the distribution point and the file-execution device.

40. A data structure (DS) for storing observable execution traces of an executable file in a behavior profile, the data structure comprising:

at least one entry (E) per system call performed by the executable file, the entry comprising an argument-to-function/method call and a timestamp of when the system call occurred.

41. The data structure according to claim 40, wherein the system call comprised in the entry is hashed, excluding the timestamp.

42. The data structure according to claim 40, wherein the argument is constituted by a classification of the argument.

43. The data structure according to claim 42, wherein the classification is a list of data elements comprising at least one of:

- an origin of the argument,
- a security label, and
- and least one argument range constraint.

Fig. 1

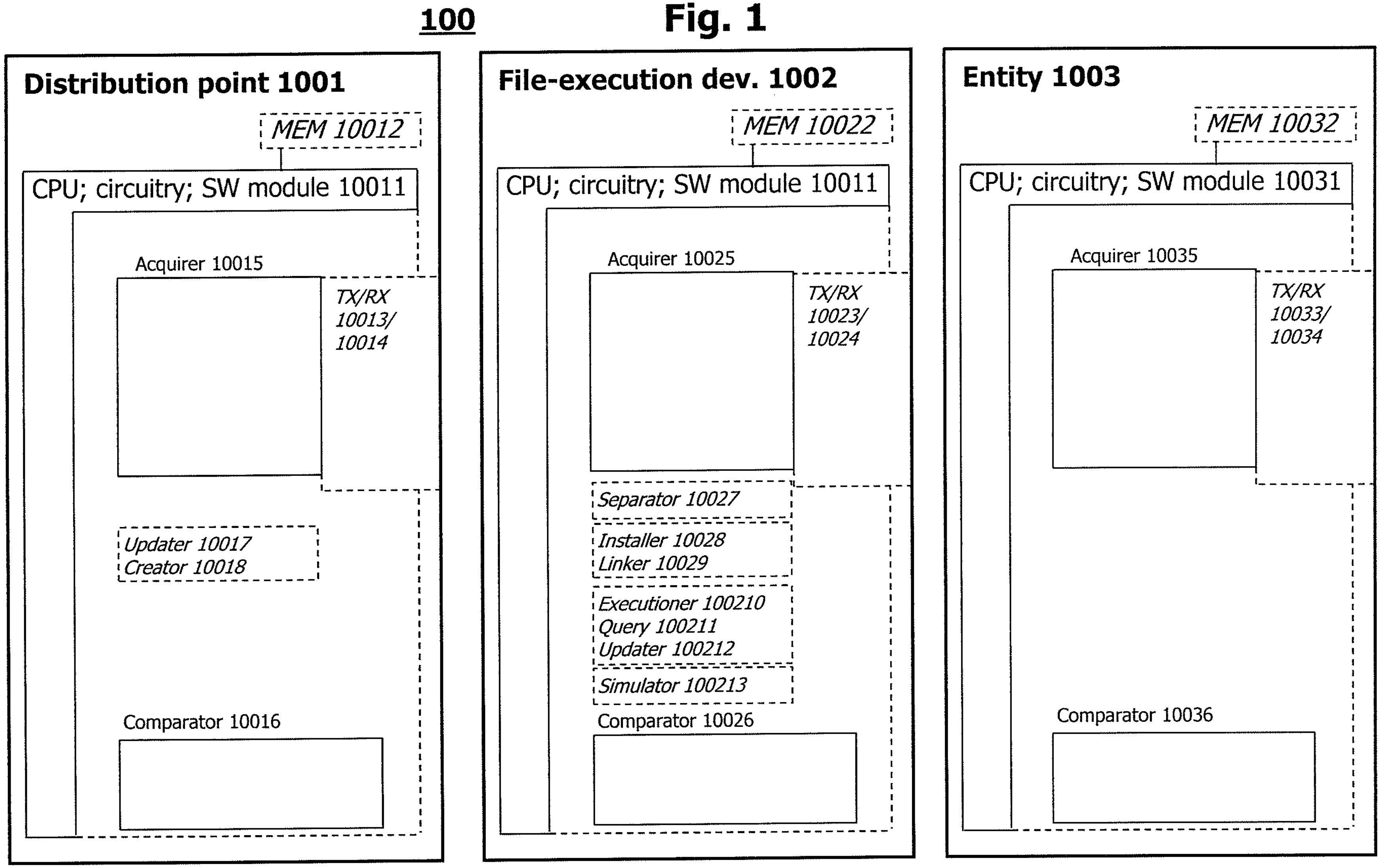

100
Distribution point 1001
MEM 10012
CPU; circuitry; SW module 10011
Acquirer 10015
TX/RX 10013/ 10014
Updater 10017
Creator 10018
Comparator 10016
File-execution dev. 1002
MEM 10022
CPU; circuitry; SW module 10011
Acquirer 10025
TX/RX 10023/ 10024
Separator 10027
Installer 10028
Linker 10029
Executioner 100210
Query 100211
Updater 100212
Simulator 100213
Comparator 10026
Entity 1003
MEM 10032
CPU; circuitry; SW module 10031
Acquirer 10035
TX/RX 10033/ 10034
Comparator 10036

Fig. 2

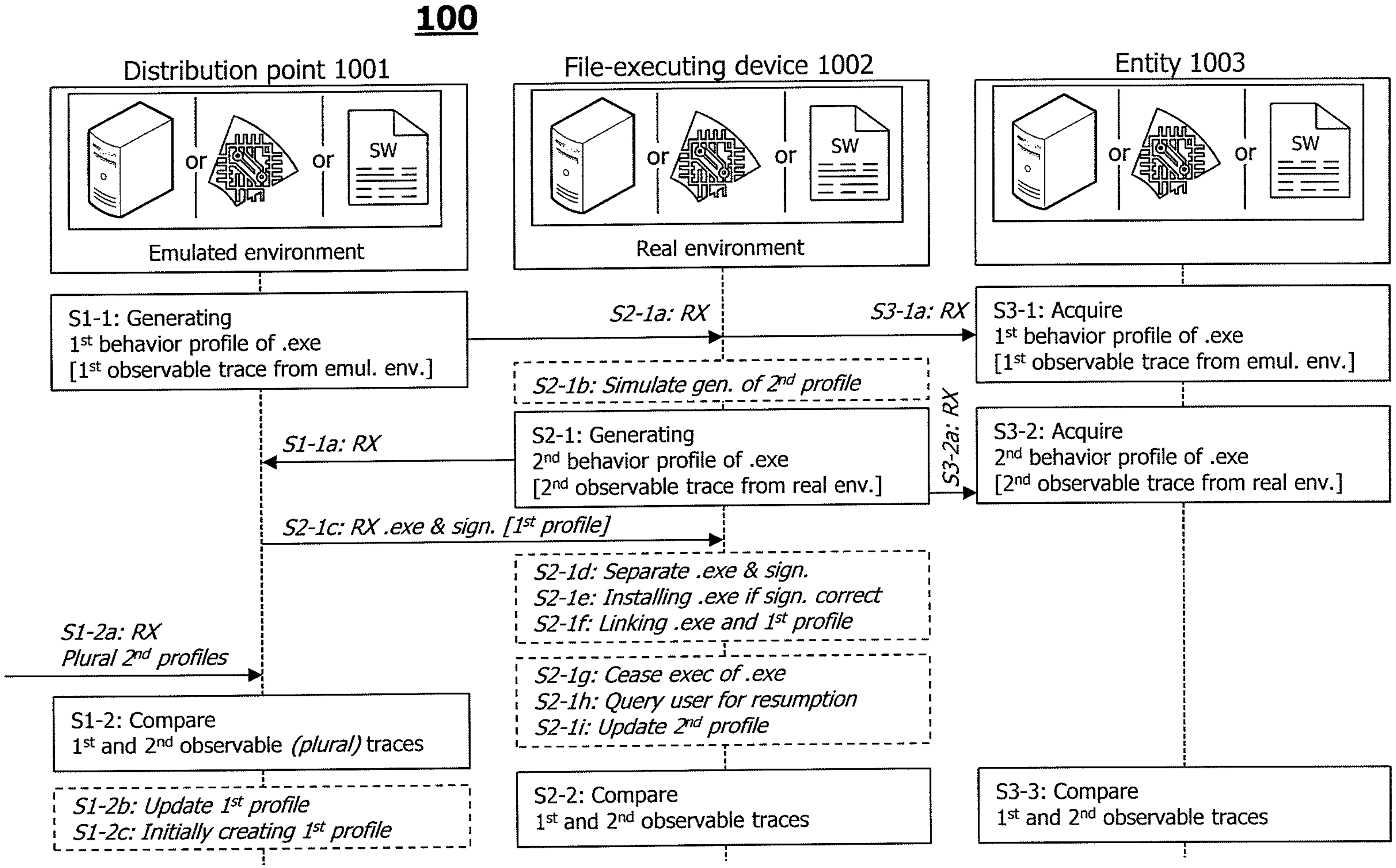
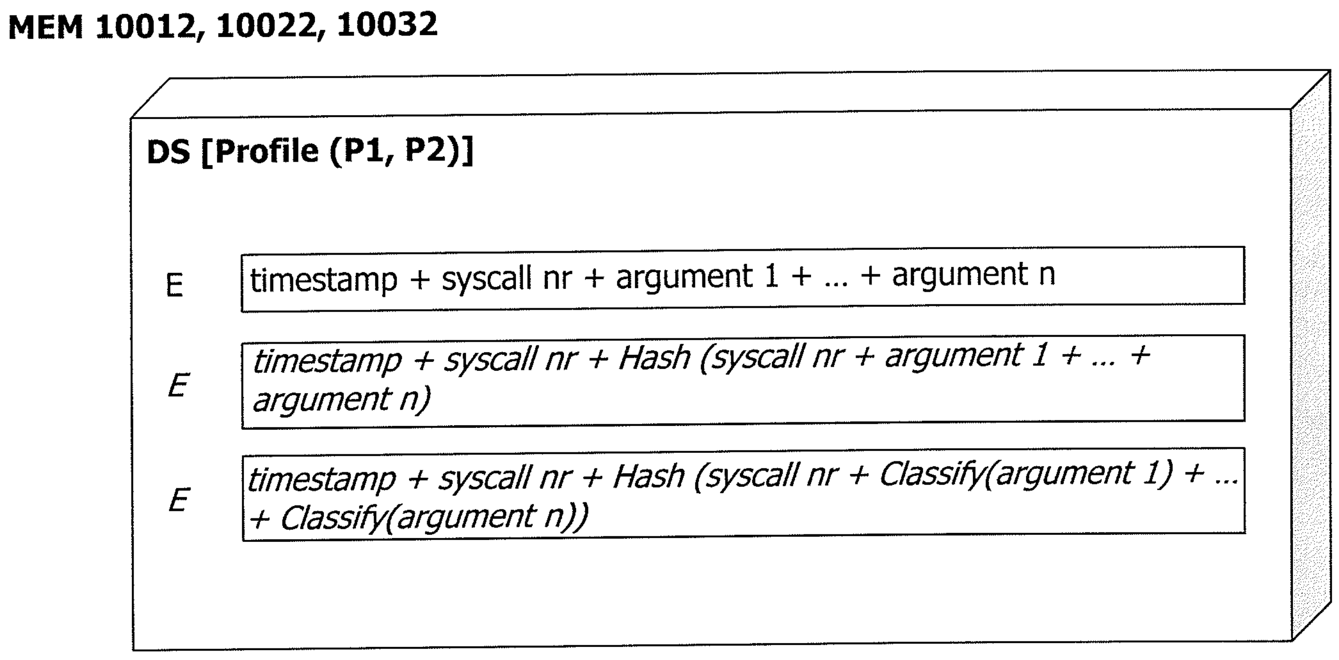

100
Distribution point 1001
or
or
SW
Emulated environment
File-executing device 1002
or
or
SW
Real environment
Entity 1003
or
or
SW
S1-1: Generating
1st behavior profile of .exe
[1st observable trace from emul. env.]
S2-1a: RX
S3-1a: RX
S3-1: Acquire
1st behavior profile of .exe
[1st observable trace from emul. env.]
S2-1b: Simulate gen. of 2nd profile
S3-2a: RX
S1-1a: RX
S2-1: Generating
2nd behavior profile of .exe
[2nd observable trace from real env.]
S3-2: Acquire
2nd behavior profile of .exe
[2nd observable trace from real env.]
S2-1c: RX .exe & sign. [1st profile]
S2-1d: Separate .exe & sign.
S2-1e: Installing .exe if sign. correct
S2-1f: Linking .exe and 1st profile
S1-2a: RX
Plural 2nd profiles
S2-1g: Cease exec of .exe
S2-1h: Query user for resumption
S2-1i: Update 2nd profile
S1-2: Compare
1st and 2nd observable (plural) traces
S1-2b: Update 1st profile
S1-2c: Initially creating 1st profile
S2-2: Compare
1st and 2nd observable traces
S3-3: Compare
1st and 2nd observable traces

Fig. 3


MEM 10012, 10022, 10032
DS [Profile (P1, P2)]
E
timestamp + syscall nr + argument 1 + ... + argument n
E
timestamp + syscall nr + Hash (syscall nr + argument 1 + ... + argument n)
E
timestamp + syscall nr + Hash (syscall nr + Classify(argument 1) + ... + Classify(argument n))

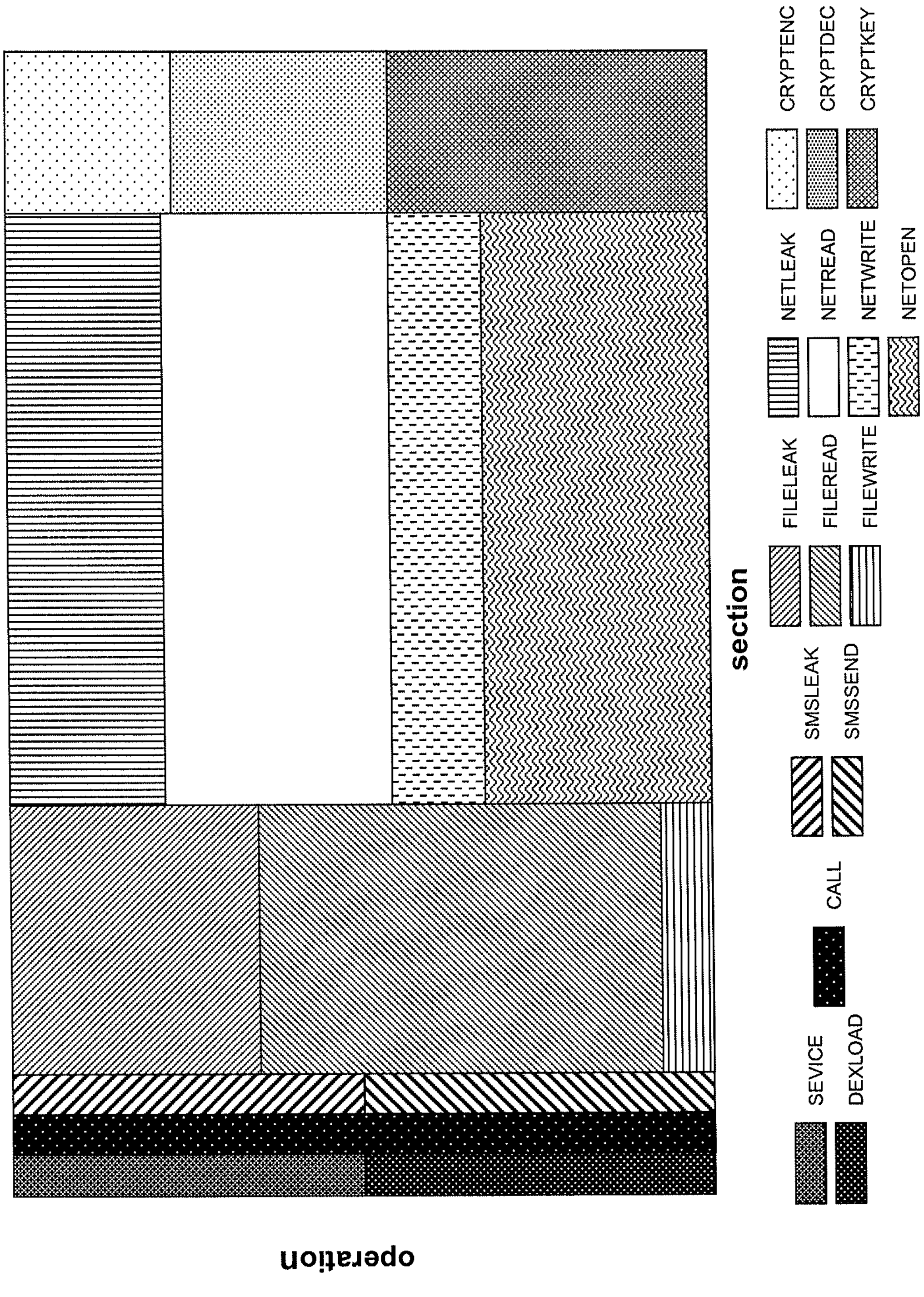

operation
section
SEVICE
DEXLOAD
CALL
SMSLEAK
SMSSEND
FILELEAK
FILEREAD
FILEWRITE
NETLEAK
NETREAD
NETWRITE
NETOPEN
CRYPTENC
CRYPTDEC
CRYPTKEY

Fig. 4A

Fig. 4B

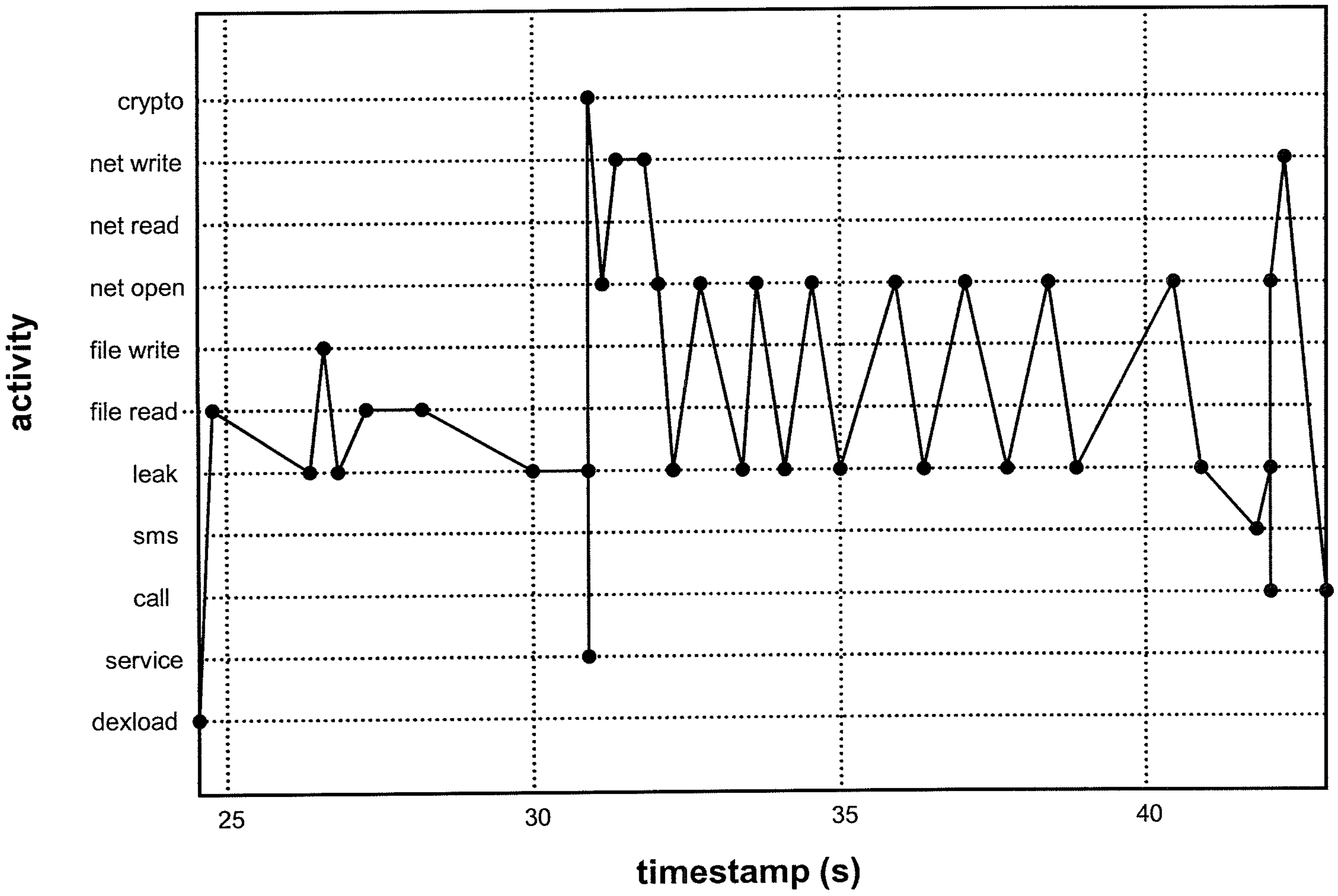
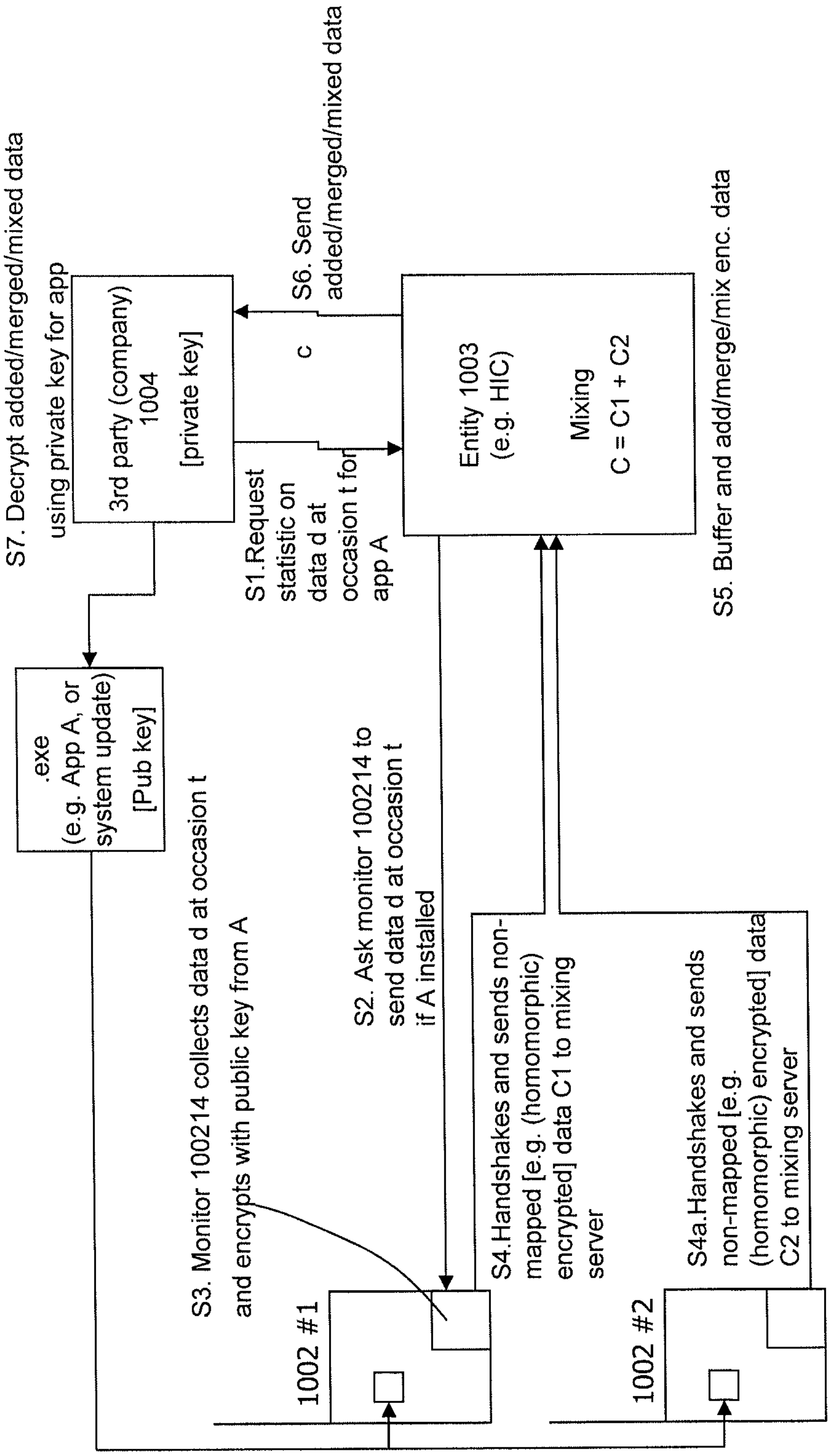

activity
crypto
net write
net read
net open
file write
file read
leak
sms
call
service
dexload
25
30
35
40
timestamp (s)

Fig. 5


S7. Decrypt added/merged/mixed data
using private key for app
3rd party (company)
1004
[private key]
S6. Send
added/merged/mixed data
c
S1.Request
statistic on
data d at
occasion t for
app A
Entity 1003
(e.g. HIC)
Mixing
C = C1 + C2
S5. Buffer and add/merge/mix enc. data
.exe
(e.g. App A, or
system update)
[Pub key]
S2. Ask monitor 100214 to
send data d at occasion t
if A installed
S3. Monitor 100214 collects data d at occasion t
and encrypts with public key from A
S4.Handshakes and sends non-
mapped [e.g. (homomorphic)
encrypted] data C1 to mixing
server
S4a.Handshakes and sends
non-mapped [e.g.
(homomorphic) encrypted] data
C2 to mixing server
1002 #1
1002 #2

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2012/076161

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/56 G06F21/62 G06F21/51
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2011/145920 A1 (MAHAFFEY KEVIN PATRICK [US] ET AL) 16 June 2011 (2011-06-16)	1-18, 23-25, 28-43
Y	paragraph [0035] - paragraph [0061] paragraph [0074] - paragraph [0078] paragraph [0094] - paragraph [0097] paragraph [0120] - paragraph [0121] paragraphs [0143], [0145] paragraph [0168] - paragraph [0176] -----	19-22, 26,27
Y	US 2007/140479 A1 (WANG JIAHE H [US] ET AL WANG JIAHE HELEN [US] ET AL) 21 June 2007 (2007-06-21) abstract paragraph [0002] - paragraph [0007] paragraph [0010] paragraph [0028] - paragraph [0034] -----	19-22, 26,27

□ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 20 February 2013	Date of mailing of the international search report 28/02/2013
Name and mailing address of the ISA/ European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Authorized officer Cartrysse, Kathy

2

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2012/076161

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011145920 A1	16-06-2011	NONE	
US 2007140479 A1	21-06-2007	NONE	