

19



OFICINA ESPAÑOLA DE
PATENTES Y MARCAS

ESPAÑA



11 Número de publicación: **2 919 085**

51 Int. Cl.:

G06F 21/56

(2013.01)

12

TRADUCCIÓN DE PATENTE EUROPEA

T3

96 Fecha de presentación y número de la solicitud europea: **04.12.2015** **E 15198039 (8)**

97 Fecha y número de publicación de la concesión europea: **06.04.2022** **EP 3029595**

54 Título: **Aparatos, dispositivos móviles, métodos y programas informáticos para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático**

30 Prioridad:

05.12.2014 DE 102014118034

45 Fecha de publicación y mención en BOPI de la traducción de la patente:
21.07.2022

73 Titular/es:

**FRAUNHOFER-GESELLSCHAFT ZUR
FÖRDERUNG DER ANGEWANDTEN
FORSCHUNG E.V. (100.0%)
Hansastr. 27c
80686 München, DE**

72 Inventor/es:

**RASTHOFER, SIEGFRIED;
MILTENBERGER, MARC y
BODDEN, ERIC**

74 Agente/Representante:

ELZABURU, S.L.P

ES 2 919 085 T3

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín Europeo de Patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre Concesión de Patentes Europeas).

DESCRIPCIÓN

Aparatos, dispositivos móviles, métodos y programas informáticos para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático

Campo técnico

- 5 Las realizaciones se refieren a aparatos, dispositivos móviles, métodos y programas informáticos para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático, más particularmente, pero no de forma exclusiva, sobre la base de uno o más puntos de registro.

Antecedentes

- 10 Esta sección introduce aspectos que pueden ser útiles para facilitar una mejor comprensión de la(s) invención(es). En consecuencia, las declaraciones de esta sección deben leerse teniendo en cuenta esto y no deben entenderse como una admisión sobre lo que se encuentra en la técnica anterior o lo que no se encuentra en la técnica anterior.

- 15 A menudo es una tarea desafiante distinguir el *malware* de las aplicaciones benignas: la ofuscación y el cifrado de cadenas a menudo hacen que los análisis estáticos sean ineficaces. Puesto que tales técnicas a menudo se pueden proporcionar como parte de un entorno de desarrollo, o como una solución de *plugin* fácil de usar, cada vez más desarrolladores pueden usar tales técnicas, para protegerse contra el robo de propiedad intelectual o para securizar sus aplicaciones.

- 20 Las aplicaciones escritas para entornos de ejecución virtuales, tales como la Máquina Virtual de Java (JVM) o la Máquina Virtual Dalvik (DVM) del sistema operativo para móviles Android, a menudo se pueden proporcionar en forma de código de bytes, que comprende instrucciones para el entorno de ejecución virtual. Dado que el código de bytes a menudo puede estar destinado a entornos virtuales bien conocidos y bien documentados, los esfuerzos para volver a traducir el código de bytes en código de programa (analizable) han avanzado considerablemente. Como resultado, los desarrolladores de aplicaciones, especialmente para el sistema operativo para móviles Android que, en muchos casos, utiliza la Máquina Virtual Dalvik, a menudo pueden tomar medidas adicionales, tales como la ofuscación, para proteger sus aplicaciones.

- 25 Dos de los enfoques principales para la ofuscación son la reflexión y el cifrado de cadenas. En la reflexión, es posible que las llamadas a subrutinas externas no se realicen directamente y que no estén contenidas explícitamente en el código del programa. En su lugar, dichas subrutinas pueden llamarse por su nombre, siendo proporcionado el nombre por una función. Tales funciones a menudo pueden comprender el cifrado de cadenas: pueden actuar como intermediarios, proporcionando valores descifrados, tales como nombres de subrutinas externas, números de teléfono, direcciones de Internet o códigos de acceso para servidores de control, para parámetros previamente especificados. En tiempo de ejecución, se puede llamar a dichas funciones para proporcionar la información necesaria, al tiempo que protegiendo a menudo la información contra el análisis estático.

- 30 Los documentos de KEVIN A ROUNDY ET AL: "Hybrid Analysis and Control of Malware", de QI XI ET AL: "An API deobfuscation method combining dynamic and static techniques" y US 2013/117855 A1 revelan métodos para el análisis y la desofuscación de código.

Compendio de realizaciones ejemplares

- 35 Se pueden aplicar algunas simplificaciones en el siguiente compendio, que pretende resaltar e introducir algunos aspectos de las diversas realizaciones ejemplificativas, pero tales simplificaciones no pretenden limitar el alcance de la(s) invención(es). En secciones posteriores se continuará con descripciones detalladas de una realización ejemplificativa preferida adecuada para permitir que aquellos con conocimientos habituales en la materia materialicen y utilicen los conceptos inventivos.

La invención se define únicamente por el alcance de las reivindicaciones adjuntas.

Realizaciones proporcionan un aparato según se define en la reivindicación independiente 1.

- 45 En algunas realizaciones, el aparato podría comprender además un módulo de extracción para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base de información relacionada con un conjunto original de instrucciones de programa, y la información relacionada con el punto o puntos de registro. Dicho módulo de extracción podría adaptarse a los requisitos del módulo de evaluación. También puede proporcionar el conjunto de instrucciones extraído para su posterior análisis en aplicaciones externas.

- 50 En al menos algunas realizaciones, el módulo de extracción puede configurarse para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base de la información relacionada con el conjunto original de instrucciones de programa. El módulo de extracción puede configurarse para determinar al menos una instrucción en el conjunto original de instrucciones de programa que accede a uno o más valores de tiempo de ejecución, y el valor o valores de tiempo de ejecución pueden basarse en la información relacionada con el punto o

- puntos de registro. El módulo de extracción puede configurarse además para extraer el conjunto extraído de instrucciones del conjunto original de instrucciones de programa en relación con el acceso al valor o valores de tiempo de ejecución. La extracción de instrucciones que acceden a uno o más valores de tiempo de ejecución puede mejorar la cobertura de código, ya que pueden omitirse instrucciones de bifurcación en algunas realizaciones, lo que puede evitar bombas de tiempo y bombas lógicas. En realizaciones, las bombas de tiempo pueden corresponderse con instrucciones de bifurcación que dependen de un tiempo o fecha, y las bombas lógicas pueden corresponderse con instrucciones de bifurcación que dependen de un activador ambiental, tal como la presencia de un entorno de depuración, o instrucciones de comando de un servidor de comando.
- En algunas realizaciones, el conjunto original de instrucciones de programa puede comprender una o más secuencias de ejecución, y la información relacionada con el conjunto de instrucciones extraído puede comprender diferentes secuencias de ejecución basadas en la secuencia o secuencias de ejecución del conjunto original de instrucciones de programa. El conjunto de instrucciones extraído puede estar basado en subconjuntos de la secuencia o secuencias de ejecución. En realizaciones, el uso de diferentes secuencias de ejecución, que pueden diferir en función de las instrucciones de bifurcación, puede mejorar la cobertura de código y evitar bombas de tiempo y bombas lógicas.
- En algunas realizaciones, el conjunto de instrucciones extraído puede comprender instrucciones para calcular y/o acceder a uno o más valores de datos y/o información de tiempo de ejecución relacionada con al menos uno del punto o puntos de registro. Limitar el alcance a la información de tiempo de ejecución relacionada con el punto o puntos de registro puede hacer que disminuyan el alcance y la complejidad de la evaluación.
- En algunas realizaciones, el módulo de extracción puede configurarse para proporcionar la información relacionada con el conjunto de instrucciones extraído sobre la base de segmentos de programa para el conjunto original de instrucciones de programa, en donde las instrucciones incluidas en el conjunto de instrucciones extraído manipulan y/o acceden a al menos un valor de datos relacionado con el punto o puntos de registro. Limitar el alcance a la información de tiempo de ejecución relacionada con el punto o puntos de registro puede hacer que disminuyan el alcance y la complejidad de la evaluación.
- En algunas realizaciones, el conjunto original de instrucciones de programa puede comprender instrucciones ofuscadas y/o valores de datos ofuscados. En algunas realizaciones, la información de tiempo de ejecución se puede usar para desofuscar las instrucciones y/o valores de datos.
- En varias realizaciones, el módulo de evaluación puede configurarse además para completar la ejecución de un conjunto de instrucciones en caso de que se produzca una excepción de ejecución. Puesto que, en realizaciones, los subconjuntos pueden determinarse sobre la base de instrucciones de bifurcación, lo que podría dar lugar a subconjuntos que podrían no ejecutarse nunca en la aplicación, continuar la ejecución con un subconjunto sucesivo puede permitir que el módulo de evaluación determine una gran parte de la información sin necesidad de entrada del usuario.
- En varias realizaciones, el módulo de evaluación puede configurarse además para reutilizar información relacionada con resultados intermedios de la ejecución de un subconjunto de instrucciones del conjunto de instrucciones extraído que comprende las mismas instrucciones en el subconjunto de instrucciones del conjunto de instrucciones extraído. Reutilizar dicha información relacionada con resultados intermedios puede hacer que disminuya el esfuerzo computacional.
- En varias realizaciones, la información de tiempo de ejecución puede corresponderse con información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro, en donde las instrucciones, parámetros, variables y/o valores de retorno de las instrucciones del conjunto original se sustituyen con instrucciones de tiempo de ejecución, parámetros de tiempo de ejecución, variables de tiempo de ejecución y/o valores de retorno de tiempo de ejecución comprendidos en la información de tiempo de ejecución. Dicha información de tiempo de ejecución se puede usar para analizar más a fondo el programa informático y para desofuscar al menos partes del programa informático.
- En algunas realizaciones, el conjunto original de instrucciones de programa puede basarse en un código de programa o un código de bytes para una máquina virtual. En varias realizaciones, el conjunto original de instrucciones de programa también puede basarse en un paquete [*bundle*] de aplicación de *software*, un Paquete de Aplicación de Android (APK), un Archivo de Java (JAR) o una aplicación de iOS. Dado que los programas informáticos para sistemas operativos para móviles, que pueden tener acceso a información o funcionalidad sensible, se lanzan en grandes cantidades, realizaciones pueden proporcionar una forma de acelerar el reconocimiento de *malware* dentro de dichos programas.
- En algunas realizaciones, el aparato puede comprender además un módulo de integración para proporcionar información relacionada con un conjunto evaluado de instrucciones, sobre la base del conjunto original de instrucciones de programa y la información de tiempo de ejecución. Tal conjunto evaluado de instrucciones podría usarse para análisis adicionales en aplicaciones externas.
- En algunas realizaciones, el módulo de integración podría configurarse para resolver llamadas de reflexión con el fin de dirigir llamadas, resolver variables, resolver parámetros, anotar intenciones [*intents*] y/o resolver transformaciones

de ofuscación para el conjunto evaluado de instrucciones, sobre la base de la información de tiempo de ejecución. La integración de dicha información de tiempo de ejecución en el conjunto evaluado de instrucciones puede facilitar análisis adicionales del programa informático.

5 En algunas realizaciones, el módulo de evaluación podría configurarse además para proporcionar información relacionada con uno o más segmentos de programa del conjunto original de instrucciones de programa, sobre la base de la información relacionada con el conjunto de instrucciones extraído. Dichos segmentos de programa podrían utilizarse para un análisis adicional en aplicaciones externas.

10 Uno de los ejemplos no limitativos describe un aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro. El aparato comprende una entrada para obtener la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro. El aparato comprende además un módulo de cálculo para determinar la información relacionada con la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro y una salida para proporcionar la información relacionada con la información de tiempo de ejecución. En realizaciones, dicho aparato, que podría estar comprendido en un dispositivo móvil o un entorno de simulación, puede proporcionar la evaluación del conjunto de instrucciones extraído para un módulo de evaluación, en un entorno de tiempo de ejecución nativo.

15 Uno de los ejemplos no limitativos describe un dispositivo móvil que comprende el aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto de instrucciones extraído e información relacionada con uno o más puntos de registro. En una de las realizaciones, un dispositivo móvil puede proporcionar un entorno de ejecución nativo para las instrucciones comprendidas en el conjunto de instrucciones extraído.

20 En algunas realizaciones, el módulo de evaluación podría comprender además una salida para proporcionar la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro a un aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro. El módulo de evaluación podría comprender además una entrada para obtener información relacionada con información de tiempo de ejecución del aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro. El módulo de evaluación podría configurarse adicionalmente para proporcionar la información de tiempo de ejecución, sobre la base de la información relacionada con información de tiempo de ejecución. En realizaciones, un aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro, que podría estar comprendido en un dispositivo móvil o un entorno de simulación, puede proporcionar la evaluación del conjunto de instrucciones extraído para un módulo de evaluación, en un entorno de tiempo de ejecución nativo.

25 Uno de los ejemplos no limitativos describe un aparato para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa. El aparato comprende una salida para proporcionar la información relacionada con el conjunto original de instrucciones de programa para un aparato para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático. El aparato comprende además una entrada para obtener información de tiempo de ejecución del aparato para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático. El aparato comprende además un módulo de evaluación para proporcionar la información relacionada con propiedades del código, sobre la base de la información de tiempo de ejecución. En realizaciones, el aparato podría usarse para proporcionar una valoración automática de riesgos y/o funcionalidades del conjunto original de instrucciones de programa.

30 Uno de los ejemplos no limitativos describe un dispositivo móvil que comprende el aparato para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa. En realizaciones, usuarios del dispositivo móvil podrían usar el aparato para valorar los riesgos de usar un programa informático no probado previamente que comprende el conjunto original de instrucciones de programa.

Realizaciones proporcionan además un método según se define en la reivindicación independiente 12.

35 Uno de los ejemplos no limitativos describe un método para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro. El método comprende obtener la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro. El método comprende además determinar la información relacionada con la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro y proporcionar la información relacionada con la información de tiempo de ejecución.

Uno de los ejemplos no limitativos describe un método para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa. El método comprende proporcionar la información relacionada con el conjunto original de instrucciones de programa para un aparato para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático. El método comprende además obtener información de tiempo de ejecución del aparato para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático y proporcionar la información relacionada con propiedades del código, sobre la base de la información de tiempo de ejecución.

Realizaciones proporcionan además un programa informático que tiene un código de programa para realizar al menos uno de los métodos anteriores, cuando el programa informático se ejecuta en un ordenador, un procesador o un componente de *hardware* programable. Otro ejemplo no limitativo es un medio de almacenamiento legible por ordenador que almacena instrucciones que, cuando son ejecutadas por un ordenador, procesador o componente de *hardware* programable, hacen que el ordenador implemente uno de los métodos descritos en este documento.

Breve descripción de las figuras

Se describirán algunas otras características o aspectos utilizando las siguientes realizaciones no limitativas de aparatos o métodos o programas informáticos o productos de programa informático solo a modo de ejemplo, y con referencia a las figuras adjuntas, en las que:

la figura 1 ilustra un diagrama de bloques de una realización de un aparato para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático;

la figura 2 ilustra un diagrama de bloques de una realización de un aparato para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático y un aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro;

la figura 3 ilustra un diagrama de bloques de una realización de un dispositivo móvil que comprende un aparato para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro;

la figura 4 muestra un fragmento de código de un código de programa ofuscado;

la figura 5 ilustra un diagrama de bloques de una estructura general de una realización;

la figura 6 ilustra un grafo de flujo de control de una muestra de código;

la figura 6a ilustra muestras de código que ilustran optimizaciones realizadas por un módulo de extracción de una realización;

la figura 6b ilustra muestras de código que ilustran instrucciones de bifurcación gestionadas por un módulo de extracción de una realización;

la figura 7 muestra un fragmento de pseudocódigo de una estructura básica y de inicialización de una realización;

la figura 8 muestra un fragmento de pseudocódigo de un algoritmo de segmentación hacia atrás de una realización;

la figura 9 muestra un fragmento de pseudocódigo de un algoritmo de segmentación de entidades llamadas de una realización;

la figura 10 muestra un fragmento de pseudocódigo de un algoritmo de segmentación de entidades llamantes de una realización;

la figura 11 muestra un fragmento de código de la familia de *malware* Pincer;

la figura 12 ilustra muestras de código que ilustran instrucciones de bifurcación gestionadas por un módulo de extracción de una realización;

la figura 13 ilustra una muestra de código que ilustra instrucciones de bifurcación gestionadas por un módulo de extracción de una realización;

la figura 14 ilustra una muestra de código que ilustra un ejemplo simplificado de una llamada de método por reflexión;

la figura 15 ilustra un diagrama de bloques de una realización de un aparato para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa;

la figura 16 ilustra un diagrama de bloques de una realización de un dispositivo móvil que comprende un aparato para

proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa;

la figura 17 ilustra un diagrama de flujo de una realización de un método para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático;

- 5 la figura 18 ilustra un diagrama de flujo de una realización de un método para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro; y

la figura 19 ilustra un diagrama de flujo de una realización de un método para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa.

10 Descripción de realizaciones

A continuación, se describirán con más detalle varias realizaciones de ejemplo con referencia a los dibujos adjuntos en los que se ilustran algunas realizaciones de ejemplo. En las figuras, los grosores de las líneas, capas o regiones pueden haberse exagerado para mayor claridad. Los componentes opcionales se ilustran con líneas de trazos, discontinuas o de puntos.

- 15 En consecuencia, aunque las realizaciones de ejemplo son susceptibles de diversas modificaciones y formas alternativas, en las figuras se muestran a modo de ejemplo realizaciones de las mismas y estas se describirán en la presente de forma detallada. Debe entenderse, sin embargo, que no hay intención de limitar las realizaciones de ejemplo a las formas particulares reveladas, sino que, por el contrario, las realizaciones de ejemplo están destinadas a cubrir todas las alternativas que se sitúan dentro del alcance de las reivindicaciones.

- 20 Los números iguales se refieren a elementos iguales o similares a lo largo de la descripción de las figuras.

Según se usa en la presente, el término "o" se refiere a una o no exclusiva, a menos que se indique lo contrario (por ejemplo, "o si no" o "o de forma alternativa"). Además, según se usa en la presente, las palabras usadas para describir una relación entre elementos deben interpretarse en un sentido amplio para incluir una relación directa o la presencia de elementos intermedios a menos que se indique lo contrario. Por ejemplo, cuando se hace referencia a un elemento como "conectado" o "acoplado" a otro elemento, el elemento puede estar directamente conectado o acoplado al otro elemento o pueden estar presentes elementos intermedios. Por el contrario, cuando se hace referencia a un elemento como "directamente conectado" o "directamente acoplado" a otro elemento, no hay presentes elementos intermedios. De modo parecido, palabras como "entre", "adyacente" y similares deben interpretarse de manera equivalente.

- 30 La terminología utilizada en este documento tiene el propósito de describir realizaciones particulares únicamente y no pretende ser una limitación de las realizaciones de ejemplo. Tal como se usan en la presente, las formas del singular "un", "una", "el" y "la" también están destinadas a incluir las formas del plural, a menos que el contexto indique claramente lo contrario. Se entenderá además que los términos "comprende", "comprendiendo", "incluye" o "incluyendo", cuando se usan en este documento, especifican la presencia de características, números enteros, pasos, operaciones, elementos o componentes mencionados, pero no excluyen la presencia o adición de otra u otras características, números enteros, pasos, operaciones, elementos, componentes o grupos de los mismos. A menos que se defina de otro modo, todos los términos (incluidos los términos técnicos y científicos) utilizados en el presente documento tienen el mismo significado que entiende comúnmente una persona con conocimientos habituales en la materia a la que pertenecen las realizaciones de ejemplo. Se entenderá además que los términos, por ejemplo, aquellos definidos en diccionarios de uso común, deben interpretarse con un significado que sea consistente con su significado en el contexto de la técnica pertinente y no se interpretarán en un sentido idealizado o demasiado formal a menos que se defina así expresamente en la presente.

- 45 Los teléfonos inteligentes se han convertido en un objetivo popular y rentable para los ataques. Las aplicaciones maliciosas pueden enviar mensajes de texto fraudulentos a números de teléfono de tarificación especial, robar los datos del usuario o integrar el teléfono en *botnets*. La identificación automática de aplicaciones de *malware* tan sofisticadas entre millones de aplicaciones disponibles puede considerarse una tarea desafiante: la ofuscación del código y el cifrado de cadenas a menudo pueden hacer que los análisis estáticos sean ineficaces. Los análisis dinámicos, por otro lado, pueden requerir entradas del usuario y pueden ser engañados con frecuencia por *malware* que detecta el entorno de ejecución emulado por la herramienta de análisis y entonces se abstiene de comportarse de manera maliciosa.

- 50 A menudo se considera que Android se ha convertido en el sistema operativo para teléfonos inteligentes más popular con una cuota dominante de instalaciones en todo el mundo. A menudo se considera que una de las razones del éxito de la plataforma es la disponibilidad de aplicaciones para casi todas las necesidades: si bien esta abundancia de aplicaciones es muy conveniente para el usuario, puede dificultar mucho la valoración de las aplicaciones debido a que puede que resulte difícil juzgar la confiabilidad y competencia del desarrollador, al igual que la propia calidad de la aplicación.

Para valorar la calidad o la seguridad de una aplicación de Android, los expertos pueden estar interesados en sus

valores de tiempo de ejecución. Este conocimiento puede considerarse importante en muchos aspectos. El analista puede querer saber, entre otras cosas, a qué Localizadores Uniformes de Recursos (URLs) se transmiten datos, a qué números de teléfono se envían mensajes SMS, cuál es el contenido de estos mensajes y qué bases de datos se leen desde el teléfono (contactos, correo electrónico, mensajes SMS, etc.). Si una aplicación, por ejemplo, abre conexiones http en lugar de https, esto podría indicar un fallo de seguridad. Si se envían mensajes SMS a números fraudulentos de tarificación especial ampliamente conocidos, esto puede indicar *malware*. Sin embargo, incluso si las aplicaciones pudieran no ser abiertamente maliciosas, muchas de ellas pueden estar ofuscadas para proteger la propiedad intelectual de sus respectivos autores, lo que dificulta la inspección manual. El código de bytes de la aplicación a menudo solo puede contener secuencias de bytes cifradas en lugar de cadenas listas para leer. El analista entonces podría investigar manualmente cómo se realizó la ofuscación de las cadenas en esta aplicación en particular y reconstruir cada una de esas cadenas.

La ofuscación también puede plantear desafíos para herramientas automatizadas de análisis estático. En el *malware* actual de Android, es posible que los métodos no se llamen directamente sino por reflexión, con el nombre del método de destino almacenado en un formato cifrado. Una llamada por reflexión puede corresponderse con especificar el nombre de una función a llamar en forma de una cadena, que puede calcularse u obtenerse de una función aparte. Es posible que la cadena concreta de destino de la llamada no se calcule antes del tiempo de ejecución, lo que puede hacer que no esté disponible para herramientas clásicas de análisis estático. Sin esta información, sin embargo, es posible que estas herramientas no generen grafos de llamadas precisos y completos, por lo que perderán información sobre qué métodos se llaman y dónde, dificultando efectivamente la detección del *malware*. Los análisis entre componentes o entre aplicaciones pueden enfrentarse a desafíos similares. En Android, la comunicación entre aplicaciones y entre componentes se puede realizar la mayoría de las veces mediante las llamadas intenciones en las que el destino se puede especificar como una cadena. Si esta cadena está ofuscada, es posible que el destinatario de la intención ya no se determine de forma estática. La disponibilidad de potentes herramientas de ofuscación puede agravar el problema, ya que las mismas podrían ayudar a difundir tales técnicas de ofuscación.

Aunque todos los valores requeridos pueden conocerse en tiempo de ejecución, las técnicas de análisis dinámico también pueden enfrentarse a varios desafíos al analizar muestras de *malware* del mundo real. Algunos enfoques, por ejemplo, pueden depender de la interceptación del tráfico de red malicioso a través de servidores *proxy* del tipo hombre en el medio o analizar la salida de un sistema de registro. Sin embargo, esto podría funcionar solo si las rutas de código que producen el tráfico de red o las salidas de registro respectivos se ejecutan realmente en tiempo de ejecución, pero es posible que ni la herramienta ni el analista sepan durante cuánto tiempo debe ejecutarse la aplicación para explorar todas las rutas. Además, el código puede comportarse de forma diferente según el entorno (emulador con respecto a teléfono real, diferentes modelos de teléfono, etc.) en el que se ejecuta. Hoy en día, las aplicaciones maliciosas pueden usar bombas de tiempo, que hacen que el *malware* se ejecute solo en un momento o fecha determinado, o bombas lógicas que activan el *malware* en función de algún activador ambiental. Esto también incluye *malware* de *botnet* que podría actuar solo en respuesta a un comando recibido de un servidor de comando y control. Con la misma importancia que lo anterior, las aplicaciones de Android pueden considerarse interactivas y es posible que se requieran ciertas interacciones del usuario para activar un comportamiento específico. Es posible que herramientas dinámicas necesiten simular estas interacciones, ya que es posible que solo se recopile información sobre rutas de código que se ejecutan realmente. En consecuencia, muchos valores de tiempo de ejecución de interés pueden seguir siendo desconocidos.

La figura 4 muestra una fracción de código del mundo real tomada de FakeInstaller, que podría considerarse una de las familias de *malware* más extendidas. La muestra se ha descompilado en código fuente de Java y se han añadido comentarios al código para facilitar la comprensión. FakeInstaller puede depender notablemente de la ofuscación para ocultar su comportamiento tanto a las herramientas de análisis como a los investigadores manuales. El ofuscador puede generar nombres de clases y métodos aleatorios, eliminando la mayor parte de la información semántica. Además, en tiempo de ejecución, en lugar de llamar a los métodos directamente, FakeInstaller puede tomar una cadena previamente cifrada y descifrarla usando una tabla de consulta. A continuación podría usar la reflexión para encontrar la clase y el método que llevan el nombre descifrado y finalmente invocar el método recuperado.

Por ejemplo, la constante de cadena VRIf3+ln9a.aTA3RYnD1BcVRV]af 4002 en la línea 6 se puede descifrar en el nombre de la clase del sistema operativo android.telephony.SmsManager que a continuación se puede cargar mediante reflexión. SMSManager podría considerarse un recurso sensible: el *malware* podría usarlo para enviar costosos mensajes SMS de tarificación especial a expensas del usuario. De hecho, la cadena BaRlta*9caBBV]a 4004 en la línea 9 se puede descifrar en sendTextMessage, el nombre del método para enviar mensajes de texto, que también se encuentra mediante reflexión. La línea 15 4006 finalmente puede invocar el método, enviando de hecho el mensaje de texto. Con la ayuda de estos valores de tiempo de ejecución, acabamos de descubrir que el método gdaBjRj es responsable de enviar mensajes de texto donde paramString1 es el número y paramString2 es el cuerpo del mensaje.

Muchas aplicaciones de *malware* actuales pueden ofuscarse de manera similar, ya sea manualmente o mediante el uso de herramientas comerciales como DexGuard. A continuación, un analista humano podría inspeccionar cuidadosamente el código de bytes descompilado, encontrar la tabla de consulta y descifrar manualmente todas las cadenas para detectar el comportamiento descrito anteriormente. Es posible que habitualmente las cadenas descifradas una vez no se reutilicen, ya que las diferentes variantes de *malware* pueden usar tablas de consulta

diferentes.

Es posible que muchas herramientas de análisis estático no puedan encontrar en absoluto la llamada a `sendTextMessage` porque es posible que no interpreten las operaciones de las cadenas ni la Interfaz de Programación de Aplicaciones (API) de reflexión y, por lo tanto, podrían carecer de la arista correspondiente del grafo de llamadas. E incluso aquellas herramientas de análisis estático que sí modelan estas APIs podrían verse engañadas por atacantes que codifican las cadenas utilizando bibliotecas de código nativo y el análisis estático no logra modelar.

Sin embargo, el código del ejemplo también puede desafiar enfoques de análisis dinámico. En primer lugar, pueden intentar encontrar una ruta de ejecución que realmente active el método `gdadbjrj`. Si, por ejemplo, el método `gdadbjrj` solo se ejecuta después de un retardo (bomba de tiempo) o después de un activador ambiental específico (bomba lógica), esto podría no ser una tarea trivial. En tales situaciones, es posible que el análisis nunca sepa cuándo es seguro detener la ejecución de la prueba dinámica y podría no acelerar fácilmente el análisis tampoco. Otro *malware* podría llamar al código malicioso solo cuando el usuario hace clic en un botón determinado. A continuación, la herramienta de análisis podría emular el clic de este botón y todas las acciones del usuario requeridas para alcanzar el estado de la interfaz de usuario que muestra el botón en primer lugar.

Varias técnicas de ofuscación para enfoques dinámicos, como mecanismos de detección de emuladores, pueden complicar aún más este análisis. La verificación en la línea 3 4008, por ejemplo, puede evitar que se ejecute el código malicioso si el entorno de ejecución muestra características de un emulador, como la presencia de ciertos archivos o un comportamiento de temporización específico. También podría abortarse si se adjunta un depurador a la aplicación. Es posible que los entornos de análisis dinámico nunca oculten por completo todas estas características y, por lo tanto, fallen con *malware* sofisticado.

Realizaciones podrían recuperar de forma totalmente automática todos los valores de tiempo de ejecución relevantes del ejemplo de la figura 4. Un analista de seguridad podría simplemente especificar las variables y/o puntos de registro para los que se deberían recuperar valores de tiempo de ejecución. En un primer paso, sin saber nada más sobre la aplicación, en este ejemplo el analista probablemente podría elegir los primeros parámetros para las llamadas de `forName` 4002, 4010 (línea 11) y `getMethod` 4006, 4012 y las dos variables `paramString1` y `paramString2` 4014 (línea 14) con el fin de obtener más información sobre las llamadas por reflexión. A continuación, un módulo de extracción puede extraer automáticamente todo el código que calcula esos valores aunque, no obstante, de manera crucial, descartando ciertos constructos de flujo de control condicional que no afectan al valor calculado. En el ejemplo, esto podría descartar la verificación de detección de emuladores en la línea 3, y también fuera de `gdadbjrj` solo retendrá exactamente el código que calcula los valores de entrada `paramString1` y `paramString2` 4014.

A continuación, un módulo de evaluación podría ejecutar solo el código reducido. Dado que se pueden eliminar verificaciones de detección de emuladores, el análisis dinámico podría ejecutar inmediatamente todas aquellas partes de `gdadbjrj` relevantes para el cálculo de los valores seleccionados. En tiempo de ejecución, el análisis puede descubrir el nombre `SmsManager.sendMessage` del método llamado por reflexión. Por último, puede notificar los números de teléfono (7151, 2858 y 9151) y cuerpos (701369431072588745752, 7012394196732588741192 y 7834194455582588771822) concretos de los mensajes SMS enviados.

Es posible que las realizaciones no requieran ninguna manipulación en el marco de Android subyacente. Pueden funcionar únicamente en el nivel de código de bytes de la aplicación de destino, a través de una transformación de código de bytes a código de bytes. Para ayudar a un mayor análisis estático del código de la aplicación, realizaciones también pueden manifestar los destinos concretos de llamadas por reflexión en las aplicaciones originales, como llamadas de método directas. Esto puede permitir un análisis más detallado utilizando herramientas de análisis estático estandarizadas, como los análisis de contaminación [taint] estática. Dado que las realizaciones pueden eliminar verificaciones de tiempo de ejecución innecesarias (p. ej., detección de emuladores) de la aplicación original, los analistas de seguridad pueden aplicar a la aplicación convertida herramientas de análisis dinámico como TaintDroid, que la aplicación original podría haber reconocido y engañado.

Las realizaciones pueden proporcionar un enfoque híbrido novedoso que puede superar las limitaciones de los análisis tanto puramente estáticos como puramente dinámicos. Las realizaciones pueden funcionar directamente en el nivel del código de bytes y podrían no requerir ningún código fuente de la aplicación bajo análisis. Las realizaciones pueden ser híbridas: primero pueden aislar estáticamente todo el código de programa que contribuye al cálculo de un valor de interés; la totalidad del resto de instrucciones de código de bytes que no influya en este valor a través de dependencias de datos podrían descartarse. Puede que el código restante sea capaz de calcular los mismos valores que el original, pero podría contener solamente las instrucciones absolutamente necesarias. Tal aislamiento de código estático puede eliminar verificaciones de flujo de control que, de lo contrario, dificultarían la ejecución dinámica, como los mecanismos de detección de emuladores. Incluso las bombas de tiempo o las bombas lógicas podrían eludirse en realizaciones. Es posible que dichas verificaciones solo influyan en el flujo de control, pero no en el cálculo del valor concreto y, por lo tanto, se eliminen. Es posible que la eliminación de las dependencias del flujo de control no afecte negativamente a la extracción de cálculos de cadenas del *malware* actual.

De manera crucial, sin embargo, el algoritmo de segmentación estático de realizaciones puede diferir en algún aspecto de algoritmos conocidos propuestos por Weiser y otros. Sus definiciones pueden ser firmes sobre la base de un

entorno fijo para la ejecución del código. Las realizaciones pueden desviarse de este concepto de firmeza generando segmentos paramétricos que emulan el comportamiento del programa original en diferentes entornos, en particular en dispositivos reales por oposición a los emuladores. El uso de una segmentación convencional requeriría un modelo preciso de cada uno de estos entornos, lo que podría no ser práctico para sistemas de Android del mundo real, por ejemplo, porque cada teléfono inteligente tiene un número diferente, cuentas diferentes, contactos diferentes, etc. Por lo tanto, las realizaciones podrían, en cambio, simular no los valores del entorno, sino las reacciones a diferentes entornos que implementa la aplicación de destino. Si una aplicación, por ejemplo, cambia su comportamiento en función de un comando de un servidor de comando y control en una *botnet*, las realizaciones pueden crear un segmento paramétrico que puede permitir que cada uno de estos comportamientos se active explícitamente sin tener que modelar la comunicación de la *botnet* como tal.

En un segundo paso, las realizaciones pueden ejecutar dinámicamente los segmentos del programa en un emulador de Android no modificado o en un teléfono Android estándar que puede permitir la reconstrucción de los valores (potencialmente cifrados) de interés. En realizaciones, el mecanismo de notificación para los valores de interés se puede incorporar instrumentalmente en los segmentos, por lo que podría no ser necesario realizar cambios en el entorno de tiempo de ejecución (emulador, *firmware* de Android, etc.). Las realizaciones pueden activar explícitamente los diferentes comportamientos del segmento paramétrico que pueden permitir la reconstrucción de los valores de interés, descifrando valores cifrados en el camino. Las realizaciones pueden incorporar instrumentalmente de manera directa el mecanismo de notificación para los valores de interés en los segmentos, por lo que podría no ser necesario realizar cambios en el entorno de tiempo de ejecución (emulador, *firmware* de Android, etc.). Dado que cada segmento se puede invocar directamente con independencia de su posición original en el código de la aplicación, es posible que no sea necesaria ninguna interacción con la interfaz de usuario u otra entrada externa durante la ejecución del segmento. Por lo tanto, las realizaciones pueden superar los problemas de cobertura de código limitada que pueden existir con los enfoques de prueba clásicos con interfaces de usuario. Los analistas de seguridad también podrían usar los segmentos calculados por realizaciones para mejorar análisis existentes.

En un tercer paso, las realizaciones podrían integrar los valores descubiertos dinámicamente de manera directa en la aplicación. Esto significa que las sentencias de código que utilizan la reflexión, por ejemplo, podrían complementarse con sentencias que llamen directamente al método de destino llamado originalmente a través de la API de reflexión. Las realizaciones pueden proporcionar muchos valores de tiempo de ejecución interesantes, tales como mensajes y direcciones de comando y control, y *malware* desofuscado con éxito que oculta llamadas de API sensibles a través de reflexiones. Además, esta inyección puede permitir que análisis estáticos estandarizados, como FlowDroid, interpreten con corrección comportamientos ofuscados anteriormente que, de otro modo, no podrían detectar sin ninguna modificación sobre la herramienta de análisis.

Las realizaciones pueden proporcionar una variación de los algoritmos de segmentación tradicionales ajustados para admitir la extracción híbrida de información de tiempo de ejecución en *malware* actualmente conocido, un sistema de ejecución dinámico para ejecutar los segmentos de código calculados y extraer los valores de interés sin interacción del usuario, y un enfoque de aumento para reinyectar los valores de datos obtenidos en la aplicación como constantes para permitir un análisis posterior con otras herramientas estándar o por parte de analistas humanos.

La figura 1 ilustra un diagrama de bloques de un aparato 10 para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático, que comprende una interfaz 12, un módulo 14 de evaluación, un módulo 16 de extracción opcional y un módulo 18 de integración opcional. La interfaz 12 está acoplada al módulo 14 de evaluación. En algunas realizaciones, la interfaz puede acoplarse además al módulo 16 de extracción opcional. En varias realizaciones, el módulo de evaluación puede acoplarse además al módulo 18 de integración opcional.

El aparato 10 comprende la interfaz 12 para obtener la información relacionada con el conjunto de instrucciones extraído e información relacionada con uno o más puntos de registro. En realizaciones, un punto de registro puede corresponderse con una combinación de una variable de interés y la instrucción en la que esta es de interés. En realizaciones, una instrucción puede corresponderse con una sentencia de programación en un lenguaje de programación de alto nivel, como Java, C, C++, Objective C ó Swift, y/o con una sentencia de programación en un lenguaje de programación de nivel máquina, como el Ensamblador. En realizaciones, una interfaz puede corresponderse con un límite compartido a través del cual dos componentes y/o módulos independientes intercambian información, como, por ejemplo, una interfaz de programación, o intercambio de información a través de un formato de archivo compartido.

En algunas realizaciones, la información de tiempo de ejecución puede corresponderse con información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro. En realizaciones, la información de tiempo de ejecución puede corresponderse con la información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro, cuando se ejecuta el conjunto original de instrucciones, y se han calculado variables. En varias realizaciones, las instrucciones, parámetros, variables y/o valores de retorno de las instrucciones del conjunto original podrían sustituirse con instrucciones de tiempo de ejecución, parámetros de tiempo de ejecución, variables de tiempo de ejecución y/o valores de retorno de tiempo de ejecución comprendidos en la información de tiempo de ejecución. En al menos algunas realizaciones, la

información de tiempo de ejecución podría usarse para el análisis de aplicaciones maliciosas, desofuscación, para análisis de código de *software* propio o de terceros, o para admitir la ingeniería inversa de protocolos, interfaces o aplicaciones, y para admitir otras herramientas de análisis de código estático o dinámico (de seguridad).

En algunas realizaciones, el conjunto original de instrucciones de programa puede basarse en un código de programa o un código de bytes para una máquina virtual. En al menos algunas realizaciones, el conjunto original de instrucciones de programa se basa en un paquete de aplicación de *software*, un Paquete de Aplicación de Android (APK), un Archivo de Java (JAR) o una aplicación de iOS. En realizaciones, un código de bytes puede corresponderse con un diseño de conjunto de instrucciones para una ejecución eficiente por parte de un entorno de *software*, p. ej. una máquina virtual, como la Máquina Virtual de Java (JVM) ó la Máquina Virtual Dalvik (DVM). En varias realizaciones, el conjunto original de instrucciones de programa puede basarse en un código de programa en un lenguaje de programación dinámico o interpretado, como p. ej. PHP (Preprocesador de Hipertexto PHP), JavaScript, Python, Ruby ó Perl.

En al menos algunas realizaciones, el conjunto original de instrucciones de programa puede comprender instrucciones ofuscadas y/o valores de datos ofuscados. En realizaciones, las instrucciones ofuscadas pueden corresponderse con instrucciones que se llaman por reflexión, p. ej. a través de la API de Reflexión de Android y/o instrucciones llamadas por el nombre, en las que el nombre se descifra en tiempo de ejecución. Los valores de datos ofuscados pueden corresponderse con valores de datos almacenados en un formato cifrado, que pueden descifrarse en tiempo de ejecución.

En al menos algunas realizaciones, el conjunto de instrucciones extraído puede comprender instrucciones para calcular y/o acceder a uno o más valores de datos y/o información de tiempo de ejecución relacionada con al menos uno del punto o puntos de registro. En alguna realización, un cálculo puede corresponderse con un cálculo aritmético o una operación lógica realizada sobre al menos el valor de datos y/o una manipulación general del valor de datos.

El aparato 10 comprende además el módulo 14 de evaluación para proporcionar la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con los puntos de registro. El módulo 14 de evaluación está configurado para proporcionar la información de tiempo de ejecución ejecutando al menos parcialmente uno o más subconjuntos de instrucciones de programa comprendidas en el conjunto de instrucciones extraído.

En al menos algunas realizaciones, la ejecución al menos parcial del subconjunto o subconjuntos de instrucciones de programa puede ser realizada por uno o más submódulos comprendidos en el módulo 14 de evaluación, o en un módulo externo, como un aparato 20 para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro como se describe en lo sucesivo. En este último caso, el módulo 14 de evaluación comprende además una funcionalidad de entrada y salida, como se describe en los siguientes párrafos, para comunicarse con el módulo externo. En realizaciones, los detalles adicionales para el módulo 14 de evaluación también pueden aplicarse al aparato 20, o, más específicamente, pero no de forma exclusiva, a un módulo 24 de cálculo.

La figura 2 ilustra un diagrama de bloques de una realización del aparato 20 para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro, que comprende una entrada 22, el módulo 24 de cálculo y una salida 26. La figura 2 ilustra adicionalmente el aparato 10, que comprende la interfaz 12, el módulo 14 de evaluación, una salida 14a y una entrada 14b, el módulo 16 de extracción opcional y el módulo 18 de integración opcional. El módulo 24 de cálculo está acoplado a la entrada 22 y la salida 26. La interfaz 12 está acoplada al módulo 14 de evaluación. En algunas realizaciones, la interfaz puede acoplarse además al módulo 16 de extracción opcional. En varias realizaciones, el módulo de evaluación puede acoplarse además al módulo 18 de integración opcional. En al menos algunas realizaciones, la salida 14a está acoplada a la entrada 22, y la salida 26 está acoplada a la entrada 14b.

El aparato 20 comprende la entrada 22 para obtener la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro. El aparato 20 comprende además el módulo 24 de cálculo para determinar la información relacionada con la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro y la salida 26 para proporcionar la información relacionada con la información de tiempo de ejecución. Las realizaciones proporcionan además un dispositivo móvil 200 que comprende el aparato 20 de la reivindicación 16, como se ilustra en la figura 3.

En las realizaciones, el módulo 14 de evaluación y/o el módulo 24 de cálculo pueden implementarse utilizando una o más unidades de procesamiento, uno o más dispositivos de procesamiento, cualesquiera medios para procesamiento, como un procesador, un ordenador o un componente de *hardware* programable que se pueda hacer funcionar con *software* adaptado en consecuencia. En otras palabras, las funciones descritas del módulo 14 de evaluación y/o el módulo 14 de cálculo también pueden implementarse en *software*, que a continuación se ejecuta en uno o más componentes de *hardware* programables. Dichos componentes de *hardware* pueden comprender un procesador de propósito general, un Procesador de Señal Digital (DSP), un microcontrolador, una Matriz de Puertas Programables

in Situ (FPGA), un procesador ARM, etc.

En algunas realizaciones, el módulo 14 de evaluación puede comprender además la salida 14a para proporcionar la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro a un aparato 20. El módulo 14 de evaluación puede comprender además la entrada 14b para obtener información relacionada con información de tiempo de ejecución del aparato 20. El módulo 14 de evaluación puede configurarse además para proporcionar la información de tiempo de ejecución, basándose en la información relacionada con información de tiempo de ejecución. En realizaciones, la información relacionada con la información de tiempo de ejecución puede corresponderse con una base de datos de valores de tiempo de ejecución, un conjunto de archivos de registro que comprenden información relacionada con valores de tiempo de ejecución, volcados de memoria de subconjuntos ejecutados del conjunto de instrucciones extraído, etc. Una entrada puede corresponderse con una interfaz para recibir información, que puede estar en valores digitales (bits) de acuerdo con un código especificado, dentro de un módulo, entre módulos o entre módulos de entidades diferentes. Una salida puede corresponderse con una interfaz para transmitir información, que puede estar representada por valores digitales (bits) de acuerdo con un código o protocolo especificado, dentro de un módulo, entre módulos o entre módulos de entidades diferentes.

En algunas realizaciones, el módulo 14 de evaluación y, de manera correspondiente, el módulo 24 de cálculo, pueden configurarse además para completar la ejecución de un conjunto de instrucciones en caso de que se produzca una excepción de ejecución. En realizaciones, una excepción de ejecución puede corresponderse con una interrupción de una ejecución debido a la falta de variables de origen, valores de retorno inesperados, errores de memoria, acceso a variables o elementos no inicializados de matrices o listas, etc.

En algunas realizaciones, el módulo 14 de evaluación y, de manera correspondiente, el módulo 24 de cálculo, pueden configurarse además para reutilizar información relacionada con resultados intermedios de la ejecución de un subconjunto de instrucciones del conjunto de instrucciones extraído que comprende las mismas instrucciones en el subconjunto de instrucciones del conjunto de instrucciones extraído. En realizaciones, los resultados intermedios podrían corresponderse con variables calculadas utilizables para el cálculo de múltiples subconjuntos.

En algunas realizaciones, el módulo 14 de evaluación puede configurarse además para proporcionar información relacionada con uno o más segmentos de programa del conjunto original de instrucciones de programa, sobre la base de la información relacionada con el conjunto de instrucciones extraído. En realizaciones, un segmento de programa podría corresponderse con un subconjunto de instrucciones que acceden a y/o manipulan uno o más valores de datos relacionados con uno o más puntos de registro.

En al menos algunas realizaciones, el aparato 10 puede comprender además un módulo 16 de extracción, para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base de información relacionada con un conjunto original de instrucciones de programa, y la información relacionada con el punto o puntos de registro.

En realizaciones, el módulo de extracción puede implementarse usando una o más unidades de procesamiento, uno o más dispositivos de procesamiento, cualesquiera medios para procesamiento, como un procesador, un ordenador o un componente de *hardware* programable que se pueda hacer funcionar con *software* adaptado en consecuencia. En otras palabras, las funciones descritas del módulo 16 de extracción también pueden implementarse en *software*, que a continuación se ejecuta en uno o más componentes de *hardware* programables. Dichos componentes de *hardware* pueden comprender un procesador de propósito general, un Procesador de Señal Digital (DSP), un microcontrolador, una Matriz de Puertas Programables in Situ (FPGA), un procesador ARM, etc.

En al menos algunas realizaciones, el módulo 16 de extracción puede configurarse para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base de la información relacionada con el conjunto original de instrucciones de programa, en el que el módulo de extracción está configurado para determinar al menos una instrucción en el conjunto original de instrucciones de programa que accede a uno o más valores de tiempo de ejecución. El valor o valores de tiempo de ejecución pueden basarse en la información relacionada con el punto o puntos de registro, y el módulo de extracción puede configurarse para extraer el conjunto extraído de instrucciones del conjunto original de instrucciones de programa en relación con el acceso al valor o valores de tiempo de ejecución. En realizaciones, el valor o valores de tiempo de ejecución pueden corresponderse con una o más variables comprendidas en uno o más grafos de dependencia de una o más variables comprendidas en el punto o puntos de registro.

En al menos algunas realizaciones, el conjunto original de instrucciones de programa puede comprender una o más secuencias de ejecución, y la información relacionada con el conjunto de instrucciones extraído puede comprender diferentes secuencias de ejecución sobre la base de la secuencia o secuencias de ejecución del conjunto original de instrucciones de programa. El conjunto de instrucciones extraído puede estar basado en subconjuntos de la secuencia o secuencias de ejecución. En realizaciones, las diferentes secuencias de ejecución pueden basarse en instrucciones de bifurcación, p. ej. dos secuencias de ejecución pueden corresponderse con dos selecciones de bifurcación diferentes de una instrucción de bifurcación. En al menos algunas realizaciones, una secuencia de ejecución puede corresponderse con una secuencia de instrucciones para calcular un valor de datos, llamar a una subrutina externa,

etc.

En algunas realizaciones, el módulo 16 de extracción puede configurarse para proporcionar la información relacionada con el conjunto de instrucciones extraído sobre la base de segmentos de programa para el conjunto original de instrucciones de programa, en donde las instrucciones comprendidas en el conjunto de instrucciones extraído pueden manipular y/o acceder a por lo menos un valor de datos relacionado con el punto o puntos de registro.

En al menos algunas realizaciones, el aparato 10 podría comprender además un módulo 18 de integración para proporcionar información relacionada con un conjunto evaluado de instrucciones, sobre la base del conjunto original de instrucciones de programa y la información de tiempo de ejecución. En al menos algunas realizaciones, el módulo 18 de integración puede configurarse para resolver llamadas de reflexión con el fin de dirigir llamadas, resolver variables, resolver parámetros, anotar intenciones y/o resolver transformaciones de ofuscación para el conjunto de instrucciones evaluado, sobre la base de la información de tiempo de ejecución. En algunas realizaciones, el conjunto de instrucciones evaluado puede corresponderse con un conjunto de instrucciones con una funcionalidad equivalente al conjunto de instrucciones original, en donde las por lo menos algunas de las variables y/o instrucciones del conjunto de instrucciones original pueden haber sido sustituidas por valores de tiempo de ejecución, para facilitar el análisis posterior del programa informático.

La figura 5 representa una arquitectura general de una realización. Un escenario de uso para al menos algunas realizaciones podría ser calcular un conjunto de valores de interés. En realizaciones, un punto de registro $\langle v, n \rangle$ puede ser un par consistente en una variable o un campo de acceso v y una sentencia arbitraria s dado que v está dentro del ámbito de s . Un valor de interés puede ser el valor concreto de la variable v en un punto de registro $\langle v, n \rangle$. Por ejemplo, un valor de tiempo de ejecución pasado a una verificación condicional es de interés, tal como $s: \text{if}(a.\text{equals}(b))$ los valores de tiempo de ejecución de a y b pueden ser, ambos, valores de interés en esta sentencia s , dando lugar a los dos puntos de registro $\langle a, s \rangle$ y $\langle b, s \rangle$. Otro ejemplo podría ser una llamada de API al método `sendMessage` tal como $s: \text{sendMessage}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}, \text{arg5})$ donde $\langle \text{arg1}, s \rangle$ y $\langle \text{arg3}, s \rangle$ podrían ser los puntos de registro en la llamada de método s . Los valores o información de tiempo de ejecución correspondientes pueden ser los valores de interés. Para ello, realizaciones pueden leer primero un archivo 5002 de APK, que puede corresponderse con el programa informático, y un archivo 5004 de configuración, que puede corresponderse con los puntos de registro, que definen los valores de las variables a recuperar junto con los puntos de código en los que se leerán. Una primera parte del análisis puede ser un cálculo 5006 de segmentación hacia atrás estática que comienza en estos puntos de código, como se explicará adicionalmente más adelante. Este paso de segmentación se puede ejecutar en un ordenador de escritorio o en un servidor de cálculo. Los segmentos, que pueden corresponderse con la información relacionada con el conjunto de instrucciones extraído, podrían a continuación usarse para construir 5008 un nuevo archivo 5010 de APK reducido que puede contener el código requerido para calcular los valores de interés, que pueden corresponderse con el conjunto extraído de instrucciones, y una actividad de ejecutor. La tarea de la actividad de ejecutor podría ser invocar los segmentos calculados y notificar los valores calculados de interés. Este nuevo archivo 5010 de APK reducido podría entonces ejecutarse 5012, por ejemplo, mediante un módulo 14 de evaluación y/o un módulo 24 de cálculo, en un emulador de Android estándar o en un teléfono real. Estos pasos podrían estar completamente automatizados; es posible que no se requiera interacción del usuario.

Opcionalmente, un analista de seguridad podría usar herramientas de análisis dinámico estandarizadas para filtrar más el APK reducido 5020. Esto puede ser beneficioso con respecto al análisis del APK 5002 original, ya que el APK reducido solo podría ejecutar el comportamiento de interés y puede quedar exento de cualquier verificación de tiempo de ejecución que de otro modo podría engañar al análisis. Por ejemplo, si un analista está interesado en posibles fugas de datos, podría usar realizaciones para producir un nuevo APK, que puede ejecutar directamente el segmento que potencialmente filtra cierta información sensible. Alternativamente, el analista podría ordenar a realizaciones que inyectasen 5016 los valores de tiempo de ejecución descubiertos 5014 en la aplicación original, lo cual podría ser proporcionado por el módulo 18 de integración. Esto puede permitir que herramientas de análisis estático estandarizadas existentes analicen aún más la aplicación 5018. Esto puede ser ventajoso con respecto al análisis de la aplicación original, ya que los valores de tiempo de ejecución y las llamadas de métodos de reflexión ahora se pueden incrustar estáticamente como constantes, respectivamente, llamadas de métodos normales, en la aplicación, lo que permite que las herramientas descubran esas llamadas y valores.

En algunas aplicaciones altamente ofuscadas, es posible que el punto de registro no se identifique directamente. Supóngase que la aplicación utiliza la reflexión para llamar a un método que envía un mensaje de texto y el analista está interesado en el número de teléfono al que se envía el mensaje de texto. En este caso, podría usar realizaciones dos veces. En la primera ronda, podría recuperar los destinos de llamadas de métodos por reflexión. Con esta información, a continuación podría identificar el punto de registro real para la segunda ronda. En otras situaciones, incluso se pueden requerir más rondas.

Calcular un valor específico de interés en una posición determinada normalmente puede requerir solo la ejecución de una pequeña fracción del programa original. Realizaciones podrían identificar y extraer solo aquellas partes del programa que realmente podrían influir en el valor de interés. La figura 6 muestra una aplicación de ejemplo simplificada, ya que el código ofuscado del mundo real puede considerarse demasiado complicado para demostrar el enfoque. En negrita, 6002, se muestran las sentencias necesarias para calcular el valor que se envía a través de

send(c) 6004; el resto podría considerarse irrelevante. Realizaciones pueden construir un nuevo método que contenga las sentencias requeridas 6002. Durante la fase de ejecución, realizaciones pueden invocar este método. Realizaciones podrían no solo eliminar la necesidad de ejecutar todas las sentencias que no están en negrita, sino que también pueden eliminar la necesidad de activar cualquier acción de interfaz de usuario específica. En su lugar, realizaciones pueden ejecutar el código relevante directamente que de otro modo estaría latente, ejecutándose solo condicionalmente sobre esas acciones.

En la segmentación tradicional según lo definido por Weiser, un segmento de programa S puede ser un programa ejecutable que se obtiene de un programa P mediante la eliminación de sentencias, de modo que S replica el comportamiento de P con respecto al criterio de segmentación. En la figura 6, los métodos 1 y 2 podrían ser irrelevantes y pueden eliminarse de forma segura ya que todos los cálculos necesarios para ello pueden ser locales para el método 3. Extraer el código respectivo del método 3 e invocarlo directamente durante la fase de ejecución posterior podría no solo reducir significativamente el tamaño del código a ejecutar y, por lo tanto, hacer que la ejecución sea más rápida, sino que también puede eliminar la necesidad de hacer clic en el botón "ClickMe". Esto puede funcionar debido a que el cálculo de un valor específico de interés en una posición determinada generalmente puede que requiera solo la ejecución de una pequeña fracción del programa original que se encuentra muy cerca del punto de registro.

La segmentación convencional puede ser insuficiente para la tarea en cuestión. Dado que el código de *malware* ofuscado del mundo real podría ser demasiado complicado para demostrar la segmentación, se presenta el ejemplo artificial 6002a de la figura 6a. En el ejemplo, el *malware* puede enviar un mensaje de texto a un número de tarificación especial en la línea 26 6008a. El número de teléfono de destino podría almacenarse en la variable number 6010a. Dependiendo del país del operador de telefonía móvil, se pueden utilizar diferentes números de teléfono 6012a (líneas 11, 15 y 19). El contenido de la variable logInfo 6014a podría no ser relevante para enviar el mensaje de tarificación especial.

El ejemplo se ha inspirado en la familia de *malware* Pincer, que puede buscar varias propiedades del sistema y abstenerse de comportarse de manera maliciosa si se encuentran rastros de un emulador. Estas verificaciones pueden formar un patrón muy común que también se puede utilizar en otras familias de *malware*. En el ejemplo, dichas verificaciones se realizan en dos lugares diferentes: Línea 5 y 10 6016a.

6004a puede mostrar la salida de un segmentador convencional para el programa de ejemplo. Se han eliminado todas las referencias a la variable logInfo no relacionada. Es posible que también se haya eliminado la primera verificación del emulador en la Línea 5 junto con todos los cálculos que solo fueron necesarios para llevar a cabo esta única verificación. Esto puede suceder porque la condición de la Línea 5 solo puede definir si se calcula el número de teléfono de destino, no cómo. Es posible que el criterio de segmentación no dependa de los datos en esta verificación. Este razonamiento puede aplicarse a muchas bombas de tiempo y bombas lógicas que, por lo tanto, podrían eliminarse de manera segura sin afectar al resultado del cálculo del valor de interés.

Cuando se ejecuta el segmento en 6004a dentro de un emulador, la variable number puede seguir siendo nula, ya que la verificación del emulador en la línea 10 puede fallar. La segmentación convencional puede apuntar a la construcción de un segmento que imite el comportamiento del programa original en el entorno actual. En este entorno actual (emulado), puede ser correcto calcular nulo para el número, pero para el analista de seguridad, que desea conocer todos los números de teléfono de destino posibles, esto podría no ser útil. En términos más generales, un analista puede desear recuperar todos los valores de interés que se pueden calcular en cualquier entorno, independientemente del entorno (típicamente emulado) en el que se ejecuta actualmente el segmento. Con un segmento tradicional, el analista aún podría emular todos estos entornos posibles. Dado que las dependencias del entorno en general, y las verificaciones del emulador en particular, pueden existir en una amplia variedad de formas, simular todas las combinaciones posibles de estas variables de entorno puede resultar inviable en la práctica.

Para hacer frente a esto, realizaciones pueden ampliar la segmentación del programa con un tratamiento especial de las condiciones. Para aquellas condiciones que pueden influir en qué valor se calcula, en tiempo de ejecución, realizaciones pueden explorar ambas bifurcaciones ya que cada bifurcación puede definir potencialmente un valor de tiempo de ejecución diferente para la respectiva variable de interés. Para forzar la ejecución de una ruta específica, realizaciones pueden sustituir todas las condiciones contenidas en el segmento con expresiones booleanas estáticas como se muestra en 6006a. Esto puede hacer que el segmento sea paramétrico con al menos una instancia de cada posible valor de interés.

Cuando se evalúa el segmento, realizaciones podrían ejecutar el ejemplo una vez para cada una de las 16 combinaciones posibles de los valores booleanos EXECUTOR_1 a EXECUTOR_4 6018a. Esto puede enumerar posibles resultados del cálculo para number.

Sin embargo, la aplicación de este principio en apps del mundo real puede generar una serie de desafíos. Cuando se sustituyen condiciones, podría ser importante no romper la semántica del programa de destino. Idealmente, para cada valor de interés calculado por realizaciones, puede existir un posible entorno de tiempo de ejecución en el que el programa original habría calculado el mismo valor.

La figura 6b 6002b muestra una realización de un cálculo más complejo para el valor de interés en el punto de registro $< x, 12 >$. Si todas las condiciones de este ejemplo se sustituyeran a ciegas, el segmento que se muestra en 6004b podría calcular el valor 1 en algún caso (EXECUTOR_0 fijado en verdadero) y no terminar en todos los demás (EXECUTOR_0 fijado en falso). Los valores correctos calculados por el programa original (16, 11, 42) podrían no ser calculados nunca por el segmento paramétrico.

Esto puede suceder porque los valores booleanos pueden permanecer constantes para cada ejecución, es decir, una instancia del segmento paramétrico puede o bien tomar siempre la bifurcación izquierda o bien tomar siempre la bifurcación derecha para la condición respectiva. En la verificación de programas, estos problemas a menudo se pueden solucionar desarrollando bucles. Dado que las realizaciones pueden preparar un segmento parametrizado para la ejecución dinámica, podría ser inviable desarrollar los bucles hasta un número suficientemente grande de iteraciones. Si el número es demasiado bajo, esto puede hacer que las realizaciones pierdan la salida real del cálculo. Si es demasiado alto, puede conducir a una explosión combinatoria de los posibles valores booleanos y rutas resultantes: para n condiciones, se podrían explorar hasta 2^n rutas.

Al menos algunas realizaciones pueden seguir un enfoque diferente. Las condiciones que solo tienen dependencias de datos en los valores de interés, es decir, son dependencias internas en el segmento, es posible que no se reemplacen con variables booleanas en primer lugar. Es posible que se conserven tal como están en el programa original. En 6002b, esto significa que las condiciones de las Líneas 5 y 6 pueden sustituirse con EXECUTOR_1 y EXECUTOR_2 respectivamente, mientras que las expresiones de la línea 3 y 8, que dependen de x , podrían mantenerse intactas, como se muestra en 6006b. De esta manera, las realizaciones podrían dirigir explícitamente la ejecución hacia rutas específicas mientras mantienen intacto el cálculo de los valores de interés en la mayoría de los casos.

En algunos casos, sustituir las condiciones originales con variables booleanas podría hacer que el segmento calculase un valor diferente al del programa original. De forma más precisa, podría haber entornos en los que el programa original calculase un valor que no pudiera ser calculado por ninguna instancia del segmento paramétrico. Especialmente para condiciones sin dependencia directa del flujo de datos en un contador de bucle, las realizaciones podrían sustituir una condición con una expresión booleana estática que podría obligar a la ejecución una vez a omitir el bucle por completo y otra a repetirse infinitamente, sin tener en cuenta la semántica original. Para evitar problemas de terminación, las realizaciones pueden limitar el recuento máximo de iteraciones a un valor fijo y abortar el bucle si se supera este límite. Aunque esto puede conducir a resultados falsos, en muchos casos, parece que no plantea problemas para analizar muestras de *malware* del estado de la técnica ya que muchos algoritmos de ofuscación y descifrado utilizados por *malware* actual podrían manipular valores del código o de un archivo. Es posible que no usen bucles para generar de manera dinámica valores completamente nuevos.

Al menos algunas realizaciones pueden sobreaproximar las rutas a ejecutar, por lo que pueden devolver falsos positivos, es decir, valores que el programa original no pudo calcular en ningún entorno dado. Dado que las realizaciones pueden no hacer suposiciones sobre el posible conjunto de entornos, también pueden explorar rutas espurias y devolver valores espurios. En la práctica, parece que la cantidad de dicha información espuria puede ser baja y no representar una carga significativa para un analista.

Al menos algunas realizaciones podrían proporcionar una gestión especial para llamadas de API que acceden a valores del entorno, como una entrada del usuario de texto libre. Dado que el segmento podría ejecutarse automáticamente sin interacción del usuario, realizaciones pueden inyectar valores ficticios en lugar de las llamadas de API reales que leen la interfaz de usuario. Esto puede evitar que los segmentos se bloqueen incluso si siguen accediendo a algunos recursos externos.

En algunas realizaciones, el algoritmo de segmentación podría seguir únicamente dependencias de datos. En el grafo de ejemplo, esto puede hacer que realizaciones excluyan el nodo 12 y todos sus predecesores del segmento. Tales dependencias de control, causadas por condiciones, pueden recibir en cambio un tratamiento especial, como se explica más adelante. Esto puede considerarse muy diferente de la segmentación tradicional, pero puede permitir que realizaciones eludan cualquier verificación de detección de emuladores que pudiera realizar la aplicación. Incluso el *malware* que oculta su malicia a través de bombas de tiempo o bombas lógicas podría analizarse con esta técnica.

La figura 7 muestra la inicialización y la estructura básica de una realización. Para cada punto de interés, realizaciones podrían crear un nuevo segmento hacia atrás, comenzando desde ese mismo punto de interés (línea 104, 7002). Dado que el punto de inicio del segmento podría no solo depender del método contenedor y sus entidades llamadas, sino también de código de inicialización realizado en métodos ejecutados anteriormente en el programa original, como el método onCreate() de la actividad contenedora, el proceso de segmentación puede continuar en todas las entidades llamantes potenciales (línea 107, 7004) del método que contiene el punto de interés. Posteriormente, realizaciones pueden insertar código para registrar dinámicamente los valores de tiempo de ejecución en los puntos de interés (línea 109, 7006). Finalmente, realizaciones pueden eliminar código que no se requiere para ejecutar el segmento (línea 114, 7008) y pueden ejecutar el segmento (línea 117, 7010).

La figura 8 muestra una versión simplificada de realizaciones que no consideran aspectos como el *aliasing*. Realizaciones pueden gestionar el lenguaje Java completo, incluidos el *aliasing*, los bucles, la recursividad, etc.

Realizaciones podrían comenzar en la sentencia de interés, por ejemplo, en `send(c)` 6004 en el ejemplo de la Figura 6 donde el valor de `c` puede ser de interés. En realizaciones, una combinación de una variable de interés y la sentencia en la que es de interés puede denominarse punto de registro.

En la línea 201 8002, realizaciones pueden inicializarse con `workList = hsend(c), {c}`. Siempre que la lista de trabajo no esté vacía, algunas realizaciones pueden seleccionar una sentencia de la misma, inicializando el análisis. Si la sentencia define o sobrescribe una variable, es posible que no se requieran valores anteriores de esta variable. Sin embargo, si el cálculo de este nuevo valor requiere otros valores, sus valores podrían calcularse antes (líneas 208 a 220 8004).

En el ejemplo, se encuentra una invocación (línea 226 8006). Es posible que sus parámetros no se marquen directamente como "requeridos", ya que en la entidad llamada es posible que no se utilicen para el cálculo particular de interés. Las posibles entidades llamadas pueden segmentarse utilizando el método `SliceCallees` (línea 225 8008) de la figura 9. Puede mapear el objeto base de la llamada (línea 306 9002) y todos los campos accesibles a través de él (línea 308 9004) en el ámbito de todas las entidades llamadas posibles. Es posible que no se mapeen parámetros ya que se trata de un análisis hacia atrás y la entidad llamada se introduce desde el sitio de retorno. A continuación, realizaciones pueden segmentar la entidad llamada respectiva (línea 312 9006) utilizando la misma función `SliceBackwards`. Cuando la entidad llamada se ha segmentado, los parámetros de la llamada (línea 318 9008), el objeto base (línea 321 9010) y los campos accesibles que no se sobrescriben en todas las entidades llamadas (Campo 323 9012) podrían volver a mapearse con la entidad llamante, como entradas requeridas. Obsérvese que realizaciones pueden considerar el método de envío como una función de biblioteca, por lo que es posible que el segmentador no elimine ningún código del mismo. En consecuencia, la función `SliceCallees` podría mantener la lista original de valores requeridos en este caso y podría planificar todos los parámetros según lo requerido en la línea 329 9014.

Para la entidad llamante, realizaciones podrían continuar con la línea 229 8010, donde los predecesores de la sentencia actual se planifican en la lista de trabajo, es decir, `a="x"` y `c=c+9`. Es posible que la primera sentencia no se marque como requerida, ya que es posible que no tenga efectos colaterales sobre ninguna de las variables de la lista requerida `variables = {c}` y el conjunto de variables simplemente se puede pasar tal cual al siguiente predecesor (`c=c+8`). La segunda sentencia define `c`, pero también podría requerir un valor para ello, lo cual marca la sentencia como necesaria, pero podría dejar la lista de variables sin cambios (línea 213 8012). La sentencia `if` añade `b` a variables (línea 215 8014). Realizaciones pueden llegar entonces a la inicialización de la variable `c` y eliminar `c` de las variables (línea 211 8016). Como `b` todavía está en la lista, la segmentación podría continuar hacia arriba usando los mismos principios. Para calcular `b`, puede ser necesario un valor para `a`. Cuando el algoritmo alcanza la definición de `a`, es posible que no se desconozcan más valores y que la lista de trabajo se quede vacía.

Para hacer frente a los bucles y la recursividad, realizaciones pueden ejecutarse en una iteración de punto fijo que guarda el conjunto de sentencias requeridas en `reqVars` junto con las variables cuyos valores aún deben determinarse mediante una segmentación hacia atrás adicional. En realizaciones, cada sentencia en el programa de destino podría o bien estar en `reqVars` una vez (es decir, es parte del segmento) o bien no estar en absoluto (es decir, no es parte del segmento). Si el segmentador encuentra la misma sentencia dos veces, debido a bucles o recursividad, es posible que ya esté en el conjunto y es posible que el segmentador no necesite procesarla adicionalmente ni procesar sus sucesores. Si todas las sentencias requeridas están en el conjunto, realizaciones podrían terminar.

Para entidades llamantes, como se muestra en la figura 10, el proceso de segmentación podría comenzar en el punto de interés y buscar hacia atrás en el método respectivo. Sin embargo, el código que se ejecuta en un momento anterior del ciclo de vida de la aplicación, como un inicializador estático o el método `onCreate()` de un componente de Android, también podría necesitar ejecutarse en tiempo de ejecución para garantizar cálculos correctos en el segmento extraído. Por lo tanto, realizaciones podrían atravesar la pila de llamadas hacia arriba para encontrar dicho código de inicialización. La técnica podría ser similar a la segmentación de entidades llamadas. Podría verificar si algún valor de parámetro (línea 405 1002), objeto base (línea 408 1004) o campo accesible a través del objeto base (línea 410 1006) puede necesitar ser conocido en el comienzo del método actual y, de ser así, podría iniciar un nuevo segmentador para que la entidad llamante encuentre su inicialización respectiva (línea 414 1008). En realizaciones, puede haber múltiples sitios de llamadas posibles para un método, y la segmentación podría iniciarse para todos ellos.

Realizaciones podrían usar una función `variablesWritten` para determinar los efectos colaterales de un método. Si un método no tiene efectos colaterales y su valor de retorno no es necesario, el sitio de llamada respectivo podría eliminarse del segmento (línea 223 8018). Esto también podría eliminar la necesidad de calcular sus parámetros, lo que puede permitir una mayor compactación de los segmentos. Realizaciones podrían permitir a expertos declarar manualmente como libres de efectos colaterales métodos de biblioteca tales como `Integer.valueOf`. Para el código de usuario, realizaciones podrían realizar un análisis de efectos colaterales automatizado en lugar de confiar en el conocimiento del dominio externo. Si todas las entidades llamadas de una sentencia sobrescriben un determinado valor, todos los valores anteriores de esta variable podrían ignorarse de forma segura y sus cálculos podrían eliminarse del seguimiento.

Si una entidad llamada nunca usa uno de sus parámetros en el segmento, el parámetro respectivo podría eliminarse para simplificar el código sin cambiar su semántica. Sin embargo, eliminar un parámetro puede violar el contrato de las interfaces o superclases implementadas. Por lo tanto, el parámetro respectivo podría mantenerse, pero

sustituyéndolo por una constante ficticia, con lo cual se evita el cálculo potencialmente costoso del valor real.

Para programas muy grandes, el cálculo de segmentos exactos podría resultar inviable. Por lo tanto, realizaciones podrían admitir cortes tanto de entidades llamadas como de entidades llamantes. Estas opciones podrían definir la profundidad máxima de la pila de llamadas hasta la cual el segmentador avanzará hacia arriba o hacia abajo desde los puntos de interés originales. Después del corte, todas las entidades llamadas adicionales podrían tomarse tal como están sin ninguna segmentación. Todas las entidades llamantes que superen el corte podrían descartarse, es decir, realizaciones podrían suponer que no se requieren más inicializaciones anteriores y podrían sustituir las variables por valores ficticios. Muchas aplicaciones pueden usar clases de API, tales como Preferencias Compartidas, para almacenar datos, que posteriormente pueden recuperarse y a continuación, por ejemplo, enviarse a internet. El almacenamiento y la recuperación podrían distribuirse entre el programa, por ejemplo ejecutándose cuando se hace clic en diferentes botones en la interfaz de usuario. Un enfoque de segmentación que podría no modelar esta dependencia de datos entre acciones del usuario podría generar un segmento incorrecto que puede que intente leer datos inexistentes de unos medios de almacenamiento de datos no inicializados. Para gestionar estos casos, realizaciones podrían resolver llamadas que escriben en almacenamiento persistente y anteponerlas al segmento.

Segmentos extraídos durante la fase de segmentación estática podrían generar un nuevo método en un archivo de APK reducido proporcionado por realizaciones, que puede corresponderse con la información relacionada con el conjunto de instrucciones extraído. Una actividad de ejecutor inyectada en el mismo archivo de APK podría llamar a todos estos métodos uno tras otro, lo que puede ocurrir después de que se haya iniciado el APL reducido en un emulador no modificado o en un teléfono Android estándar. Un ejecutor, que puede corresponderse con un módulo de evaluación o un aparato 20, podría escribir los valores de tiempo de ejecución calculados en una base de datos SQLite en la tarjeta SD del dispositivo que a continuación podría descargarse y evaluarse en un ordenador de escritorio. Dado que los segmentos podrían ejecutarse directamente, independientemente de su posición original en el código de la aplicación, realizaciones podrían no requerir interacción del usuario que, de lo contrario, podría ser necesaria para acceder a la ubicación del código de las sentencias informáticas. Si, por ejemplo, el código extraído originalmente estaba contenido en un manejador de clics de botón, esto podría haber requerido que el usuario o un controlador de prueba automatizado hiciera clic en ese botón para ejecutarlo. Realizaciones podrían ejecutar el código segmentado directamente, haciendo que esto sea innecesario. Es posible que la aplicación reducida ni siquiera contenga elementos de la Interfaz Gráfica de Usuario (GUI) de la aplicación original. La aplicación reducida también podría estar empaquetada con los mismos recursos que la aplicación original, de modo que el código que podría cargar cadenas cifradas, por ejemplo, desde recursos externos, podría encontrar esos recursos también en el APK reducido.

Realizaciones podrían tratar con código de detección de emuladores que puede intentar eludir el análisis: dado que este código no influye directamente en los valores de interés, realizaciones podrían convertirlo en parte del segmento. La figura 11 muestra un fragmento de código de la familia de *malware* Pincer que busca varias propiedades del sistema y se abstiene de comportarse de forma maliciosa si se encuentran rastros de un emulador. Esto puede considerarse un patrón muy común que también puede usarse en otras familias de *malware*. En el ejemplo, el *malware* envía una cadena a un número específico a través de SMS. Se podría considerar que el contenido del texto variable en la línea 13 1102 no depende de los datos en las verificaciones del emulador 1104 (Líneas 6-11), lo que podría permitir que las verificaciones se eliminen de forma segura. Lo mismo podría aplicarse también a las bombas de tiempo y las bombas lógicas. Por la misma razón, es posible que el código respectivo tampoco se convierta en parte del segmento.

Si las bifurcaciones condicionales sí influyen en el valor de interés, como, por ejemplo, para x en la línea 10 1202 del ejemplo elaborado de la figura 12, realizaciones podrían explorar ambas bifurcaciones. Cada bifurcación podría definir potencialmente diferentes valores de tiempo de ejecución para la variable respectiva. Para forzar la ejecución de una ruta específica, realizaciones podrían sustituir todas las condiciones contenidas en el segmento por expresiones booleanas estáticas que están bajo el control del ejecutor. El ejecutor podría ejecutar el ejemplo una vez con la totalidad de las cuatro combinaciones posibles de las variables booleanas EXECUTOR_0 1204a y EXECUTOR_1 1204b (verdadero/verdadero, verdadero/falso, falso/verdadero, falso/falso).

Cuando se sustituyen condiciones, podría ser importante no romper la semántica del programa de destino. El código 1208 podría calcular el mismo valor para la variable x que el código original 1206. Por lo tanto, las variables que solo dependen de los valores de interés, es decir, pueden ser dependencias internas en el segmento, podrían no transformarse. En 1206, esto significa que las condiciones de las Líneas 3 1210a y 4 1212b podrían sustituirse por EJECUTOR_0 y EJECUTOR_1 respectivamente, mientras que las expresiones en la línea 2 1212 y 6 1214, que dependen de x, podrían mantenerse intactas. Esta distinción podría usarse para evitar generar resultados espurios debido a una bifurcación inconsistente. De esta manera, realizaciones podrían dirigir explícitamente la ejecución hacia rutas específicas mientras mantienen intactos muchos algoritmos y evitan cambios en la semántica del algoritmo, por ejemplo, para rutinas de descifrado.

Realizaciones también podrían hacer frente a la carga de código dinámica y a métodos nativos, siempre que todo el código que contiene los puntos de registro pueda estar contenido dentro del código de bytes del APK en el momento de la instrumentación. Si, por ejemplo, el valor de un mensaje SMS se calcula llamando a una función cargada dinámicamente usando la reflexión, o invocando un método nativo, el segmentador podría declarar esta función como requerida y el módulo de evaluación podría ejecutar la función como cualquier otra función, proporcionando la misma

implementación que también podría invocarse durante la ejecución normal de la aplicación.

Si los valores en cuestión dependen de condiciones, como en 1206, los segmentos de código contener más de una ruta posible. Para obtener todos los valores posibles, realizaciones podrían seguir rutas posibles y podrían evaluar valores respectivos en esas rutas. En general, esto podría conducir a 2^n rutas si n es el número de condiciones entre la introducción de la variable y la posición donde se utiliza. En la práctica, sin embargo, muchas de esas rutas podrían generar el mismo valor. Por lo tanto, realizaciones podrían admitir la selección aleatoria de un número máximo predefinido de instancias de segmento (es decir, combinaciones de las variables booleanas) para ejecutar. Si bien esto generalmente puede conducir a valores de interés perdidos, en la aplicación práctica este punto podría no ser una limitación.

En realizaciones, no todas las bifurcaciones condicionales podrían afectar a los valores de interés. Por lo tanto, realizaciones podrían aplicar un conjunto de heurísticas, haciendo que se ejecute realmente solo una fracción configurable de estas rutas. Si, por ejemplo, solo se pudiera alcanzar un punto de registro si una variable tiene un valor constante específico, como en la Línea 5 1302 de la figura 13, realizaciones podrían suponer que la variable tiene este valor constante. La bifurcación "else" de dicha condición podría no ejecutarse, ya que podría no conducir a ningún punto de registro. La eliminación de condiciones con bifurcaciones vacías podría considerarse otra mejora. En la Línea 1, ninguna de las bifurcaciones podría influir en el valor de interés x y y , respectivamente, ambas bifurcaciones podrían eliminarse durante la segmentación. Esto podría permitir eliminar también toda la condición, ya que puede volverse irrelevante qué bifurcación se elige.

En realizaciones, solo para aquellas condiciones para las que ambas bifurcaciones influyen en el valor de interés, se podrían ejecutar las dos rutas posibles. Para limitar el tiempo de ejecución requerido en esos casos, realizaciones podrían omitir rutas. Realizaciones podrían preferir abandonar rutas que están más alejadas del punto de registro, ya que se podría suponer que tienen menos influencia en el resultado real del cálculo.

Los enfoques de análisis estático existentes pueden basarse en un grafo de llamadas para determinar el método de destino cuya ejecución es provocada realmente por la invocación de un método. Para la gran parte de aplicaciones de *malware* que se ofuscan mediante llamadas de métodos de reflexión, como el ejemplo de la figura 4, la construcción de un grafo de llamadas puede fallar. Es posible que algunas herramientas no admitan en absoluto llamadas por reflexión durante la construcción del grafo de llamadas, mientras que otras pueden admitir la resolución de llamadas por reflexión con cadenas de destino constantes, pero es posible que no gestionen nombres de clases o métodos contruidos dinámicamente. Sin embargo, realizaciones podrían ayudar a esas herramientas estandarizadas al manifestar los valores de tiempo de ejecución de destinos de llamadas por reflexión resueltos durante la ejecución dinámica como llamadas de método ordinarias en el código de bytes de la aplicación. Esto podría permitir que algoritmos de construcción de grafos de llamadas existentes construyesen un grafo de llamadas firme con facilidad.

El método `obfuscated()` en la figura 14 muestra un ejemplo simplificado de una llamada de método por reflexión. Realizaciones podrían detectar dos posibles valores de tiempo de ejecución diferentes para la variable t , a saber, "foo" y "bar". Realizaciones podrían sustituir el método ofuscado con el método `directCalls` que puede contener aristas de llamadas directas a estos dos métodos de destino. Para permitir casos en los que hay más destinos de llamadas que algunas realizaciones no detectaron dinámicamente, la antigua llamada por reflexión podría conservarse en la bifurcación de caída (línea 10 1402). Herramientas de análisis estandarizadas podrían entonces analizar el archivo de APK enriquecido sin requerir una gestión especial para operaciones de cadenas o de reflexión utilizadas para crear el nombre del método de destino. Los archivos de APK enriquecidos podrían ser funcionalmente equivalentes a sus respectivos originales y podrían usar solo código de nivel de aplicación normal. Es posible que ejecutarlos no requiera ningún cambio en el sistema operativo o el emulador.

Realizaciones podrían usar esta técnica para llamadas de métodos de reflexión, pero también se podría aplicar para otras cadenas ofuscadas, como números de teléfono de destino de mensajes SMS, para ayudar a herramientas existentes de detección de *malware* basadas en patrones. Una de las realizaciones también podría proporcionar la inyección de las cadenas de acción y los Identificadores Uniformes de Recursos (URI) de intenciones de Android utilizadas para la comunicación entre componentes y entre aplicaciones. Muchos análisis estáticos podrían fallar si estas cadenas no son constantes, en la medida en que ya no pueden mapear emisores y receptores de intenciones. Lo mismo también puede aplicarse a cadenas de nombres de clase que se usan con intenciones explícitas. Si solo se descifran en tiempo de ejecución, los análisis estáticos solo podrían suponer de forma conservadora que todos los destinatarios son posibles, lo que puede considerarse muy impreciso. Inyectar estas cadenas como constantes podría permitir que herramientas reconstruyesen el grafo de llamadas entre componentes con mayor precisión e identificasen correctamente los flujos de datos entre componentes y aplicaciones que de otro modo no serían posibles.

Algunas realizaciones se han implementado en un proyecto HARVESTER. En HARVESTER, la segmentación estática se podría implementar utilizando el marco Soot para el análisis y la transformación estáticos de programas, que puede leer y escribir directamente archivos de APK de Android. HARVESTER podría usar la representación intermedia Jimple de Soot, que puede comprender un lenguaje plano de tres operandos optimizado para análisis estáticos.

Para que los segmentos generados sean lo más precisos (y, por lo tanto, lo más pequeños) posible, HARVESTER puede requerir un grafo de llamadas preciso de la aplicación de destino. El motor de construcción de grafos de

llamadas de Soot, Spark, puede proporcionar dicho grafo de llamadas inicial. Soot puede ser capaz de construir un grafo de llamadas sobre la base de puntos de entrada de un programa. Dado que tal punto de entrada podría no existir para aplicaciones de Android, se podría crear un método principal artificial utilizando el componente AndroidEntryPointCreator de FlowDroid, una herramienta existente de análisis estático de contaminación estática de código abierto basada en Soot.

Inicialmente, es posible que a este grafo le falten aristas de llamada para llamadas por reflexión, ya que HARVESTER debe descubrir exactamente esas llamadas durante su pasada dinámica. Una vez que las llamadas han sido descubiertas e incrustadas como llamadas directas en el APK, se podrían realizar iteraciones del mismo procedimiento, expandiendo el grafo de llamadas, descubriendo potencialmente más sitios de llamadas de reflexión, resolviendo dinámicamente sus destinos, etc. Sin embargo, en la práctica, el proyecto HARVESTER descubrió que esta reiteración podría no ser necesaria, ya que el *malware* actual podría no llamar habitualmente a las propias APIs de reflexión mediante reflexión.

HARVESTER puede ejecutar los segmentos extraídos llamando a sus puntos de entrada desde una actividad de ejecutor artificial inyectada en el archivo de APK resultante. Sin embargo, un segmento puede tener referencias a otros componentes de Android. Además, si el código se ejecutó originalmente después de que el usuario hiciera clic en un botón, puede esperar que se inicie la actividad de hospedaje. HARVESTER podría así emular el ciclo de vida de esta actividad en tiempo de ejecución. Sin embargo, las llamadas ingenuas [*naive*] a métodos del ciclo de vida podrían causar problemas, ya que el sistema operativo Android puede esperar que se fijen ciertas variables internas, tales como el Contexto utilizado, lo cual no se puede lograr a través de la interfaz normal. HARVESTER, por lo tanto, podría usar la reflexión para inyectar estos valores de inicialización. La forma habitual de cambiar actividades a través de Intenciones puede no presentar una alternativa adecuada, ya que podría no darle a HARVESTER el control sobre los métodos del ciclo de vida, lo que llevaría a una ejecución de código innecesaria.

Realizaciones proporcionan además un aparato 30 para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa. La figura 15 ilustra un diagrama de bloques del aparato 30, que comprende una salida 32, una entrada 34 y un módulo 36 de evaluación. La figura 15 ilustra además un aparato 10. La salida 32 está acoplada al aparato 10, que además está acoplado a la entrada 34. La entrada 34 está además acoplada al módulo 36 de evaluación. El aparato 30 comprende la salida 32 para proporcionar la información relacionada con el conjunto original de instrucciones de programa para el aparato 10 como se ha descrito anteriormente. El aparato 30 comprende además la entrada 34 para obtener información de tiempo de ejecución del aparato 10 y el módulo 36 de evaluación para proporcionar la información relacionada con propiedades del código, sobre la base de la información de tiempo de ejecución.

En realizaciones, la información relacionada con propiedades del código puede corresponderse con información relacionada con indicaciones de que el conjunto original de instrucciones de programa comprende *malware*, información relacionada con subrutinas externas llamadas por el conjunto original de instrucciones de programa, información relacionada con fuentes de datos utilizadas por el conjunto original de instrucciones de programa, información relacionada con datos transmitidos por el conjunto original de instrucciones de programa, etc.

En realizaciones, el módulo 36 de evaluación puede implementarse usando una o más unidades de procesamiento, uno o más dispositivos de procesamiento, cualesquiera medios para procesamiento, como un procesador, un ordenador o un componente de *hardware* programable que se pueda hacer funcionar con *software* correspondientemente adaptado. En otras palabras, las funciones descritas del módulo 36 de evaluación también pueden implementarse en *software*, que a continuación se ejecuta en uno o más componentes de *hardware* programables. Dichos componentes de *hardware* pueden comprender un procesador de propósito general, un Procesador de Señal Digital (DSP), un microcontrolador, un procesador ARM, etc.

La figura 16 ilustra un diagrama de bloques de un dispositivo móvil 300 que comprende el aparato 30. En realizaciones, un dispositivo móvil, o transceptor móvil, puede corresponderse con un teléfono inteligente, un teléfono celular, un equipo de usuario, un equipo de radiocomunicaciones, un móvil, una estación móvil, un ordenador portátil, un ordenador portátil de tipo *notebook*, un ordenador personal, un Asistente Digital Personal (PDA), un lápiz de memoria de Bus Serie Universal (USB), un automóvil, un transceptor de retransmisión móvil para comunicación D2D, etc. También se puede hacer referencia a un transceptor móvil como Equipo de Usuario (UE) o móvil en línea con la terminología del 3GPP.

La figura 17 ilustra un diagrama de flujo de una realización de un método para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático. El método comprende obtener 42 la información relacionada con el conjunto de instrucciones extraído e información relacionada con uno o más puntos de registro y proporcionar 44 la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con los puntos de registro. La provisión de la información de tiempo de ejecución se basa en ejecutar al menos parcialmente uno o más subconjuntos de instrucciones de programa comprendidas en el conjunto de instrucciones extraído.

La figura 18 ilustra un diagrama de flujo de realizaciones de un método para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de

instrucciones e información relacionada con uno o más puntos de registro. El método comprende obtener 52 la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro. El método comprende además determinar 54 la información relacionada con la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro. El método comprende además proporcionar 56 la información relacionada con la información de tiempo de ejecución.

La figura 19 ilustra un diagrama de flujo de realizaciones de un método para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa. El método comprende proporcionar 62 la información relacionada con el conjunto original de instrucciones de programa para un aparato 10 como se ha descrito anteriormente. El método comprende además obtener 64 información de tiempo de ejecución del aparato 10 y proporcionar 66 la información relacionada con propiedades del código, basándose en la información de tiempo de ejecución.

Algunas realizaciones comprenden un circuito de control digital instalado dentro del aparato para llevar a cabo el método. Un circuito de control digital de este tipo, p. ej. un Procesador de Señal Digital (DSP), debe programarse en de manera correspondiente. Por lo tanto, aún otras realizaciones también proporcionan un programa informático que tiene un código de programa para llevar a cabo realizaciones del método, cuando el programa informático se ejecuta en un ordenador, un procesador digital o un componente de *hardware* programable. Una de las realizaciones adicionales es un medio de almacenamiento legible por ordenador que almacena instrucciones que, cuando son ejecutadas por un ordenador, procesador o componente de *hardware* programable, hacen que el ordenador implemente uno de los métodos descritos en este documento.

Una persona experta en la técnica reconocerá fácilmente que pasos de los diversos métodos descritos anteriormente pueden llevarse a cabo mediante ordenadores programados. Aquí, algunas realizaciones también están destinadas a cubrir dispositivos de almacenamiento de programas, por ejemplo, medios de almacenamiento de datos digitales, que son legibles por máquina o por ordenador y codifican programas de instrucciones ejecutables por máquina o ejecutables por ordenador donde dichas instrucciones llevan a cabo algunos o todos los pasos de métodos descritos en este documento. Los dispositivos de almacenamiento de programas pueden ser, por ejemplo, memorias digitales, medios de almacenamiento magnético tales como discos magnéticos y cintas magnéticas, unidades de disco duro o medios de almacenamiento de datos digitales legibles ópticamente. Las realizaciones también están destinadas a cubrir ordenadores programados para llevar a cabo dichos pasos de métodos descritos en el presente documento o matrices lógicas programables (in situ) ((F)PLAs) o matrices de puertas programables (in situ) ((F)PGAs), programadas para llevar a cabo dichos pasos de los métodos antes descritos.

La descripción y los dibujos simplemente ilustran los principios de la invención. Por lo tanto, se apreciará que aquellos expertos en la técnica serán capaces de idear varias disposiciones que, aunque no se describen o muestran explícitamente en este documento, incorporan los principios de la invención y están incluidas dentro de su espíritu y alcance. Además, todos los ejemplos citados en este documento están destinados principalmente y de manera expresa solo a fines pedagógicos para ayudar al lector a comprender los principios de la invención y los conceptos a los que contribuye el inventor o inventores para promover la técnica, y deben interpretarse de manera que no se limiten a dichos ejemplos y condiciones mencionados específicamente. Además, todas las declaraciones en este documento que mencionan principios, aspectos y realizaciones de la invención, así como ejemplos específicos de la misma, pretenden abarcar sus equivalentes.

Los bloques funcionales denominados "medios para..." (realizar una determinada función) se entenderán como bloques funcionales que comprenden circuitería que está adaptada para realizar una determinada función, respectivamente. Por lo tanto, "medios para algo" también puede entenderse como "medios que están adaptados o son adecuados para algo". Unos medios que están adaptados para realizar una determinada función, por lo tanto, no implican que dichos medios estén necesariamente realizando dicha función (en un instante de tiempo dado).

Las funciones de los diversos elementos que se muestran en las figuras, incluidos los bloques funcionales etiquetados como "medios", "medios de control", "medios de transmisión", "medios de recepción", "medios de transmisión/recepción", "medios de procesamiento", etc., pueden proporcionarse mediante el uso de *hardware* dedicado, tal como "un controlador", "un transmisor", "un receptor", "un transceptor", "un procesador", etc., así como *hardware* capaz de ejecutar *software* en asociación con *software* apropiado. Además, cualquier entidad aquí descrita como "medios" puede corresponderse con o implementarse como "uno o más módulos", "uno o más dispositivos", "una o más unidades", etc. Cuando las proporciona un procesador, las funciones pueden ser proporcionadas por un solo procesador dedicado, por un solo procesador compartido o por una pluralidad de procesadores individuales, algunos de los cuales pueden ser compartidos. Además, el uso explícito del término "procesador" o "controlador" no debe interpretarse como una referencia exclusiva a *hardware* capaz de ejecutar *software*, y puede incluir implícitamente, entre otros, *hardware* de procesador de señal digital (DSP), procesador de red, circuito integrado de aplicación específica (ASIC), matriz de puertas programables in situ (FPGA), memoria de solo lectura (ROM) para almacenar *software*, memoria de acceso aleatorio (RAM) y medios de almacenamiento no volátil. También se puede incluir otro *hardware*, convencional o personalizado. Su función puede llevarse a cabo a través del funcionamiento de lógica del programa, a través de lógica dedicada, a través de la interacción de control del programa y lógica dedicada, o incluso manualmente, siendo seleccionable la técnica particular por el implementador según se entienda más

específicamente a partir del contexto.

En una evaluación del proyecto HARVESTER, se ha medido la cobertura de puntos de registro. Una herramienta perfecta alcanzaría todos los puntos de registro. Se eligieron trece muestras diferentes de *malware*, algunas de las cuales se sabe que presentan una alta ofuscación (FakeInstaller, GinMaster y Obad). Estas muestras pueden depender notablemente de la reflexión para enmascarar los destinos de llamadas de métodos. Es posible que se sepa que otra familia de *malware*, Pincer, dificulta la evaluación dinámica a través de técnicas antiemulación. Ssuel y Dougalek roban varios elementos de datos privados.

En resumen, se ha demostrado que HARVESTER detecta al menos un valor para el 99% de todos los puntos de registro. Debido a la ejecución controlada de bifurcaciones que influyen en valores, HARVESTER puede tener éxito en la notificación de valores múltiples para muchos puntos de registro. Descubrir más de un valor puede ser útil para analizar el comportamiento de una aplicación (por ejemplo, números de teléfono fraudulentos para fraude por SMS en diferentes países en lugar de simplemente un solo número). HARVESTER podría incluso tener la capacidad de hacer frente a las técnicas de antianálisis utilizadas por la familia de *malware* Pincer, donde puede extraer con éxito el número y mensaje de SMS, URLs, comandos de *shell* y varios accesos a archivos.

En realizaciones, la pequeña fracción de puntos de registro perdidos puede venir provocada principalmente por la falta de inicialización de valores de tiempo de ejecución importantes debido a cortes configurados durante el análisis estático.

Harvester también se comparó con SAAF, un enfoque estático para identificar valores de parámetros sobre la base de un enfoque de segmentación hacia atrás a partir de una llamada de método. Este método puede considerarse similar a la parte de análisis estático hacia atrás en HARVESTER. SAAF y HARVESTER se evaluaron sobre 2,600 muestras de *malware* de MobileSandbox. Los puntos de registro para ambas herramientas fueron el número y el mensaje correspondiente de mensajes de texto. Los resultados de SAAF muestran que es posible que la herramienta no admita semántica de operaciones de cadenas, tales como la concatenación. En lugar de la cadena concatenada, SAAF puede notificar los dos operandos distintos. Esto puede dar solo una idea parcial del comportamiento de la aplicación. En algunos casos, es posible que SAAF ni siquiera encuentre todos los fragmentos necesarios del número de teléfono de destino (por ejemplo, 1065-5021-80133). Por el contrario, HARVESTER extrajo los números de SMS finales y completos para todas las muestras e incluso notifica números en los casos en los que SAAF no proporcionó ningún dato.

Además, es posible que SAAF no admita la extracción de los textos de los mensajes SMS que se envían, ya que generalmente no son constantes de cadena, sino que se construyen a través de concatenación y la transformación de cadenas. Debido a su naturaleza estática, es posible que SAAF tampoco gestione llamadas por reflexión, lo que podría no ser un problema para HARVESTER debido a la ejecución dinámica. Esto muestra que realizaciones como HARVESTER pueden tener la oportunidad de gestionar operaciones semánticas de manera más efectiva que las puramente estáticas como SAAF.

Como ya muestran los resultados anteriores, HARVESTER puede mostrar un mayor *recall* que enfoques clásicos basados en pruebas actualmente conocidos en la bibliografía, que, en promedio, solo pueden alcanzar una cobertura de código del 30% al 60%. Bajo el supuesto de que los puntos de registro se distribuyen uniformemente en el código de la aplicación, esto significa que solo se podría llegar si acaso a entre el 30% y el 60% de todos los puntos de registro. Estos números sugieren que un enfoque híbrido de segmentación y ejecución podría ser más apropiado para aplicaciones de Android que encontrar patrones de prueba concretos.

Para validar aún más esta impresión, HARVESTER también se probó en 150 muestras de 18 familias de *malware* tomadas del Malware Genome Project. El *recall* de HARVESTER se comparó con el de Monkey de Google, que se ejecutó con al menos 1,000 eventos generados aleatoriamente por aplicación que se limitaron a interacciones normales del usuario (hacer clic, deslizar, usar el botón de navegación). El objetivo era encontrar los números de teléfono a los que se envían mensajes SMS. Para contar los respectivos puntos de registro alcanzados por Monkey, se incorporó instrumentalmente el código de bytes de las muestras de *malware* para crear una entrada de registro directamente antes de enviar el mensaje. Los resultados se evaluaron utilizando Logcat. Todas las pruebas con Monkey se realizaron en un emulador de Android 4.1 (API versión 16).

Como era de esperar, las técnicas de detección de emuladores impidieron que Monkey alcanzara nunca ningún punto de registro en la mayoría de las muestras de *malware*. En total, Monkey solo encontró valores para el 15.6% de todos los puntos de registro. En solo el 9.33% de todas las aplicaciones, encontró un valor para al menos un punto de registro. Muchas aplicaciones maliciosas, tales como las familias de *malware* GoldDream, BaseBridge y BgServ, así como la aplicación DogWars, pueden realizar sus actividades maliciosas en un servicio en segundo plano que se pone en marcha cuando se reinicia el teléfono. Para obtener los valores de tiempo de ejecución respectivos, enfoques dinámicos tradicionales también podrían generar dichos eventos externos y reiniciar el teléfono. HARVESTER podría, en cambio, ejecutar directamente los segmentos de código que contienen los puntos de registro y, por lo tanto, es posible que no necesite emular estos eventos. Además, herramientas como Monkey solo podrían mejorar la cobertura de código al activar interacciones en la interfaz de usuario. Sin embargo, algunas aplicaciones de *malware* de la familia GPSSMSpy contienen un receptor de anuncios que filtra directamente mensajes SMS entrantes y que puede ser

completamente distinto de la interfaz de usuario. Si bien es posible que Monkey nunca ejecute el código respectivo, HARVESTER puede invocar directamente el segmento que contiene la fuga de datos independientemente de su posición original en el código.

Para superar estos problemas con Monkey, se utilizó el kit de herramientas de pruebas dinámicas de código abierto AndroidHooker, que primero puede preparar el emulador con "datos personales de usuario" falsos, tales como contactos, antes de instalar la aplicación y ejercitarla con Monkey. AndroidHooker también puede enviar eventos externos, como mensajes SMS entrantes, y podría reiniciar el emulador durante la prueba para activar acciones que solo suceden en el momento del arranque. Este enfoque puede revelar el mensaje SMS especial en algunas aplicaciones, pero es posible que no resuelva el problema de la cobertura de código en general. Por ejemplo, aún puede fallar si el código malicioso solo se ejecuta después de recibir un comando de un servidor remoto, como, por ejemplo, en la familia de *malware* GoldDream. No obstante, HARVESTER puede tener éxito ya que la verificación condicional del comando del servidor podría no ser parte del segmento que calcula HARVESTER, y el código que contiene el punto de registro podría ejecutarse de manera directa e incondicional. Debido a estos problemas, AndroidHooker solo encontró el 16.31% de todos los puntos de registro. En el 10.67% de todas las aplicaciones, encontró un valor para al menos un punto de registro.

En definitiva, al encontrar valores para el 74.47% de todos los puntos de registro, HARVESTER puede mostrar un *recall* mucho mejor para estas muestras de *malware*, ya que es posible que no esté limitado ni por bombas de tiempo ni por bombas lógicas, y es posible que no requiera ninguna entrada externa para su simulación. En el 86% de todas las aplicaciones, se encontró un valor para al menos un punto de registro. Para aumentar aún más estos números, en realizaciones, se puede considerar en el análisis la comunicación entre componentes.

Las tiendas de aplicaciones, como Google Play Store, pueden recibir muchos miles de aplicaciones de Android nuevas o actualizadas por día, las cuales pueden ser verificadas en busca de un comportamiento malicioso. Por lo tanto, se requieren herramientas rápidas que se adapten a escala al tamaño del mercado. HARVESTER se probó en 10,282 muestras de *malware* de VirusShare, 620 aplicaciones (solo troyanos de SMS) de Malware Genome Project y 2,600 muestras de *malware* de MobileSandbox. HARVESTER se configuró con 3 puntos de registro tanto para los números de teléfono de SMS como para los mensajes de texto respectivos. 1,926 de todas las aplicaciones bajo evaluación contenían al menos un punto de registro en cada una de estas dos categorías. Con HARVESTER, los valores reales de números de teléfono y mensajes de texto se pueden encontrar de manera efectiva, por lo que se podrían comparar con listas negras conocidas o aplicar filtros existentes para identificar *malware* fraudulento.

Las evaluaciones de rendimiento notificadas en esta sección se ejecutaron en un servidor de cálculo con 12 núcleos AMD Opteron 8356 que ejecutan Debian Linux 2.6 con una VM de Servidor de 64 bits HotSpot Java de Oracle versión 1.7.0 y un tamaño de almacenamiento dinámico máximo de 20 GB para evitar la recolección de basura intermedia. En promedio, HARVESTER tardó entre 25 y 45 segundos y extrajo varios números de teléfono de SMS distintos y mensajes SMS distintos. Por lo tanto, HARVESTER podría usarse para análisis masivos y al mismo tiempo entregar resultados muy rápidamente.

Durante el análisis, se han utilizado realizaciones, como el proyecto HARVESTER, para encontrar y desofuscar una serie de técnicas de ofuscación. Por ejemplo, un número creciente de aplicaciones sofisticadas de *malware* para Android, como Obad, pueden usar la reflexión para llamar a métodos identificados por constantes de cadenas cifradas que solo se descifran en tiempo de ejecución. HARVESTER se utilizó para recuperar los destinos de estas llamadas de métodos por reflexión y encontró dos patrones de ofuscación populares. En el primer patrón, solo podrían ofuscarse llamadas de API sensibles, como "getSubscriberID", "getDeviceId" o "sendTextMessage", lo que probablemente sea el resultado de una ofuscación manual para obstaculizar a los analistas humanos o herramientas automáticas que buscan llamadas de API sensibles como CHABADA. En el segundo patrón, pueden ofuscarse todas las llamadas de métodos, incluso las que no son críticas, como "StringBuffer.append()" o "String.indexOf()", lo que con la mayor probabilidad sea el resultado de herramientas de ofuscación automáticas como DexGuard. En algunas aplicaciones, incluso las propias llamadas por reflexión se llamaron nuevamente a través de reflexión para producir una ofuscación de múltiples etapas. El ejemplo de la figura 4 puede comprender una ofuscación de múltiples etapas del tipo mencionado, que puede ser muy difícil de entender para un analista manual. HARVESTER puede extraer el método llamado así como los valores de parámetros concretos de la invocación en estos casos.

HARVESTER extrajo además muchos números de tarificación especial distintos de varias familias conocidas de *malware* troyano de SMS como "Pincer". Se pueden encontrar muchos números en varias muestras, lo que los convierte en buenos candidatos para su inclusión en listas negras. Además, algunos troyanos de SMS almacenan el número de mensajes enviados en SharedPreferences, un almacenamiento de claves-valores proporcionado por el marco de trabajo de Android. HARVESTER puede encontrar muchas claves como "SENDED SMS COUNTER KEY" o "sendCount" utilizadas para este propósito. Algunas muestras incluso pueden usar claves como "cost" para almacenar la cantidad total de dinero robado hasta el momento. Basándose en estos valores, un *malware* puede decidir cuándo se envía el siguiente mensaje SMS de tarificación especial. Harvester también encontró aplicaciones que contactan con un servidor de comando y control (C&C) a través de SMS con comandos como "40659+3079+4128302+x+a" ó "3079+4128302+x+a". Dado que los mismos comandos pueden volver a aparecer en varias muestras, también podrían usarse para la inclusión en listas negras.

HARVESTER también puede extraer los Localizadores Uniformes de Recursos (URLs) concretos de solicitudes del Protocolo de Transferencia de Hipertexto (HTTP) enviadas por aplicaciones. Estos URLs pueden dar pistas sobre si una aplicación es maliciosa o no. Harvester puede extraer no solo conexiones a servidores de anuncios, sino también muchos URLs de servidores de C&C bien conocidos. Además, también puede extraer muchos URIs locales de teléfonos interesantes para acceder a proveedores de contenido, como "content://sms", "content://mms" o "tel://<number>" que pueden ser utilizados por *malware* para leer mensajes SMS/MMS ó iniciar llamadas telefónicas sin conocimiento del usuario.

HARVESTER también se puede usar para extraer valores de tiempo de ejecución para métodos de API de ejecución de comandos como Runtime.exec(). Las aplicaciones que contienen los comandos su y chmod pueden considerarse probablemente *exploits* a nivel de raíz [*root*]. HARVESTER puede detectar dichos comandos incluso en caso de ofuscación.

Algunas aplicaciones benignas pueden cifrar datos sensibles, como conversaciones de chat o información de tarjetas de crédito, antes de almacenarlos localmente en el teléfono. Sin embargo, este cifrado puede resultar inútil si se utiliza la misma clave simétrica codificada de manera fija para todas las instalaciones de la aplicación, como, por ejemplo, WhatsApp, de la cual HARVESTER extrajo la clave de cifrado obteniendo los valores pasados al constructor de la clase SecretKeySpec.

En análisis posteriores, se demostró que la información de tiempo de ejecución y/o el conjunto de instrucciones evaluado, como los resultados de HARVESTER, pueden mejorar los resultados de otras herramientas. Comparación del *recall* del rastreador de flujo de datos estáticos FlowDroid en aplicaciones de *malware* del mundo real antes y después de inyectar valores de tiempo de ejecución para llamadas de métodos por reflexión con HARVESTER. En la muestra ofuscada original, FlowDroid detectó solo 9 fugas distintas. Después de usar HARVESTER con la opción de sustituir llamadas por reflexión por sus respectivas entidades llamadas concretas, FlowDroid detectó 26 fugas de privacidad, casi tres veces más que antes. Estas 26 fugas incluyeron el robo de la IMEI ó el número de teléfono a través de SMS, lo cual se ofuscó a través de reflexión.

Las herramientas dinámicas a menudo pueden padecer una cobertura de código limitada si una aplicación depende de entradas del usuario. Con los segmentos de código extraídos por HARVESTER, las herramientas dinámicas pueden ejecutar directamente el código de interés, independientemente de su posición original en el programa original. Es posible que no se requiera interacción del usuario con la aplicación, lo que elimina problemas de cobertura de código con los enfoques existentes de generación de entradas.

Para evaluar cómo HARVESTER puede mejorar la precisión y el *recall* de herramientas existentes en aplicaciones ofuscadas, FlowDroid y TaintDroid se probaron en diez aplicaciones seleccionadas al azar de DroidBench, que se ofuscaron con DexGuard. Todas las llamadas a métodos de API se sustituyeron por llamadas por reflexión en cadenas cifradas. Los resultados muestran que FlowDroid inicialmente no pudo detectar ninguna fuga en las aplicaciones ofuscadas. Después de desofuscar las aplicaciones con HARVESTER a través de la inyección de valores de tiempo de ejecución, FlowDroid encontró las mismas fugas que en la versión original no ofuscada.

Aquellos expertos en la materia deben apreciar que cualquier diagrama de bloques de este documento representa vistas conceptuales de circuitería ilustrativa que incorpora los principios de la invención. De manera similar, se apreciará que cualquier diagrama de flujo, flujograma, diagrama de transición de estados, pseudocódigo y similares representa varios procesos que pueden representarse sustancialmente en un medio legible por ordenador y así ejecutarse por medio de un ordenador o procesador, independientemente de que dicho ordenador o procesador se muestre o no explícitamente.

Debe señalarse además que los métodos revelados en la memoria descriptiva o en las reivindicaciones pueden implementarse mediante un dispositivo que tenga medios para llevar a cabo cada uno de los pasos respectivos de estos métodos.

REIVINDICACIONES

1. Un aparato (10) para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático, extrayéndose el conjunto extraído de instrucciones de un conjunto original de instrucciones de programa del programa informático, comprendiendo el aparato
 - 5 una interfaz (12) para obtener información relacionada con el conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro, siendo un punto de registro una combinación de una variable de interés y una instrucción en la que esta es de interés, en donde la información relacionada con el conjunto de instrucciones extraído se basa en el conjunto original de instrucciones de programa y en el punto o puntos de registro;
 - 10 un módulo (14) de evaluación para proporcionar la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con los puntos de registro, ejecutando al menos parcialmente uno o más subconjuntos de instrucciones de programa comprendidas en el conjunto extraído de instrucciones,
 - 15 en donde la información de tiempo de ejecución se corresponde con información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro, en donde las instrucciones, parámetros, variables y/o valores de retorno contenidos en las instrucciones del conjunto original de instrucciones de programa se sustituyen con instrucciones de tiempo de ejecución, parámetros de tiempo de ejecución, variables de tiempo de ejecución y/o valores de retorno de tiempo de ejecución dentro de la información de tiempo de ejecución; y
 - 20 un módulo (18) de integración para proporcionar un conjunto evaluado de instrucciones, sobre la base del conjunto original de instrucciones de programa y la información de tiempo de ejecución, correspondiéndose el conjunto evaluado de instrucciones con un conjunto de instrucciones con una funcionalidad equivalente al conjunto original de instrucciones, en donde al menos algunas de las variables y/o instrucciones del conjunto original de instrucciones de programa han sido sustituidas por valores de tiempo de ejecución, para facilitar un análisis adicional del programa informático.
- 25 2. El aparato (10) de la reivindicación 1 que comprende además un módulo (16) de extracción, para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base del conjunto original de instrucciones de programa, y la información relacionada con el punto o puntos de registro.
- 30 3. El aparato (10) de la reivindicación 2, en donde el módulo (16) de extracción está configurado para proporcionar la información relacionada con el conjunto de instrucciones extraído, sobre la base del conjunto original de instrucciones de programa, en donde el módulo de extracción está configurado para determinar al menos una instrucción en el conjunto original de instrucciones de programa que accede a uno o más valores de tiempo de ejecución, y en donde el valor o valores de tiempo de ejecución se basan en la información relacionada con el punto o puntos de registro, y en donde el módulo de extracción está configurado para extraer el conjunto extraído de instrucciones del conjunto original de instrucciones de programa en relación con el acceso al valor o valores de tiempo de ejecución.
- 35 4. El aparato (10) de la reivindicación 2, en donde el conjunto original de instrucciones de programa comprende una o más secuencias de ejecución, y en donde la información relacionada con el conjunto de instrucciones extraído comprende diferentes secuencias de ejecución basadas en la secuencia o secuencias de ejecución del conjunto original de instrucciones de programa, y/o en el que el conjunto de instrucciones extraído se basa en subconjuntos de la secuencia o secuencias de ejecución.
- 40 5. El aparato (10) de la reivindicación 1, en donde el conjunto extraído de instrucciones comprende instrucciones para calcular y/o acceder a uno o más valores de datos y/o información de tiempo de ejecución relacionada con al menos uno del punto o puntos de registro.
- 45 6. El aparato (10) de la reivindicación 2, en donde el módulo (16) de extracción está configurado para proporcionar la información relacionada con el conjunto de instrucciones extraído sobre la base de segmentos de programa para el conjunto original de instrucciones de programa, en donde las instrucciones comprendidas en el conjunto extraído de instrucciones manipulan y/o acceden a por lo menos un valor de datos relacionado con el punto o puntos de registro.
- 50 7. El aparato (10) de la reivindicación 1, en donde el conjunto original de instrucciones de programa se basa en un código de programa o un código de bytes para una máquina virtual, o en donde el conjunto original de instrucciones de programa se basa en un paquete de aplicación de *software*, un Paquete de Aplicación de Android (APK), un Archivo de Java (JAR) o una aplicación de iOS.
8. El aparato (10) de la reivindicación 1, en donde el módulo (18) de integración está configurado para resolver llamadas de reflexión con el fin de dirigir llamadas, resolver variables, resolver parámetros, anotar intenciones y/o resolver transformaciones de ofuscación para el conjunto de instrucciones evaluado, sobre la base de la información de tiempo de ejecución.
- 55 9. El aparato (10) de la reivindicación 1, en donde el módulo (14) de evaluación está configurado además para

proporcionar información relacionada con uno o más segmentos de programa del conjunto original de instrucciones de programa, sobre la base de la información relacionada con el conjunto de instrucciones extraído.

10. Un sistema que comprende el aparato (10) para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático de la reivindicación 1 y un aparato (20) para proporcionar información relacionada con información de tiempo de ejecución, sobre la base de información relacionada con un conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro, en el que la información relacionada con el conjunto de instrucciones extraído se basa en un conjunto original de instrucciones de programa y se basa en el punto o puntos de registro, siendo un punto de registro una combinación de una variable de interés y una instrucción en la que esta es de interés, comprendiendo el aparato (20)
 - 5 una entrada (22) para obtener la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro del aparato (10) para evaluar información de tiempo de ejecución;
 - un módulo (24) de cálculo para determinar la información relacionada con la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con el punto o puntos de registro; y
 - 15 una salida (26) para proporcionar la información relacionada con la información de tiempo de ejecución al aparato (10) para evaluar información de tiempo de ejecución, en donde la información de tiempo de ejecución se corresponde con información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro, en donde las instrucciones, parámetros, variables y/o valores de retorno contenidos en las instrucciones del conjunto original de instrucciones de programa se sustituyen con instrucciones de tiempo de ejecución, parámetros de tiempo de ejecución, variables de tiempo de ejecución y/o valores de retorno de tiempo de ejecución dentro de la información de tiempo de ejecución.
11. Un sistema que comprende el aparato (10) para evaluar información de tiempo de ejecución de un conjunto extraído de instrucciones sobre la base de al menos una parte de un programa informático de la reivindicación 1 y un aparato (30) para proporcionar información relacionada con propiedades del código, sobre la base de un conjunto original de instrucciones de programa de un programa informático, comprendiendo el aparato (30)
 - 25 una salida (32) para proporcionar información relacionada con el conjunto original de instrucciones de programa para el aparato (10) para evaluar la información de tiempo de ejecución;
 - una entrada (34) para obtener la información de tiempo de ejecución del aparato (10) para evaluar la información de tiempo de ejecución;
 - 30 un módulo (36) de evaluación para proporcionar la información relacionada con propiedades del código, sobre la base de la información de tiempo de ejecución, de manera que la información relacionada con las propiedades del código comprende una de información relacionada con indicaciones de que el conjunto original de instrucciones de programa comprende *malware*, información relacionada con subrutinas externas llamadas por el conjunto original de instrucciones de programa, información relacionada con fuentes de datos utilizadas por el conjunto original de instrucciones de programa e información relacionada con datos transmitidos por el conjunto original de instrucciones de programa.
12. Un método para evaluar información de tiempo de ejecución de un conjunto de instrucciones extraído sobre la base de al menos una parte de un programa informático, extrayéndose el conjunto de instrucciones extraído de un conjunto original de instrucciones de programa del programa informático, comprendiendo el método
 - 40 obtener (42) información relacionada con el conjunto extraído de instrucciones e información relacionada con uno o más puntos de registro, siendo un punto de registro una combinación de una variable de interés y una instrucción en la que esta es de interés, en donde la información relacionada con el conjunto extraído de instrucciones se basa en el conjunto original de instrucciones de programa y se basa en el punto o puntos de registro; y
 - 45 proporcionar (44) la información de tiempo de ejecución, sobre la base de la información relacionada con el conjunto de instrucciones extraído y la información relacionada con los puntos de registro, ejecutando al menos parcialmente uno o más subconjuntos de instrucciones de programa comprendidas en el conjunto de instrucciones extraído,
 - en donde la información de tiempo de ejecución se corresponde con información relacionada con instrucciones, parámetros y/o variables comprendidos en el conjunto original de instrucciones de programa y relacionada con el punto o puntos de registro, en donde las instrucciones, parámetros, variables y/o valores de retorno contenidos en las instrucciones del conjunto original de instrucciones de programa se sustituyen con instrucciones de tiempo de ejecución, parámetros de tiempo de ejecución, variables de tiempo de ejecución y/o valores de retorno de tiempo de ejecución dentro de la información de tiempo de ejecución, y
 - 50 proporcionar un conjunto evaluado de instrucciones, sobre la base del conjunto original de instrucciones de programa y la información de tiempo de ejecución, correspondiéndose el conjunto evaluado de instrucciones con un conjunto
 - 55

de instrucciones con una funcionalidad equivalente al conjunto original de instrucciones, en el que al menos algunas de las variables y/o instrucciones del conjunto original de instrucciones de programa han sido sustituidas por valores de tiempo de ejecución, para facilitar un análisis adicional del programa informático.

- 5 13. Un programa informático que tiene un código de programa para llevar a cabo el método de la reivindicación 12 cuando el programa informático se ejecuta en un ordenador, un procesador o un componente de *hardware* programable.

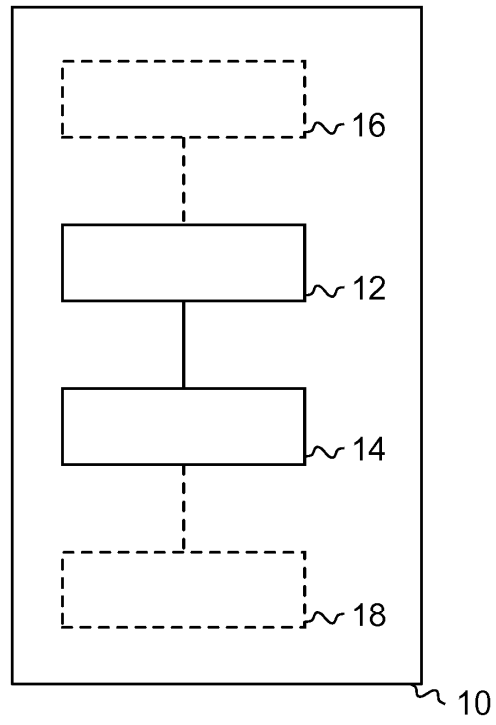


Fig. 1

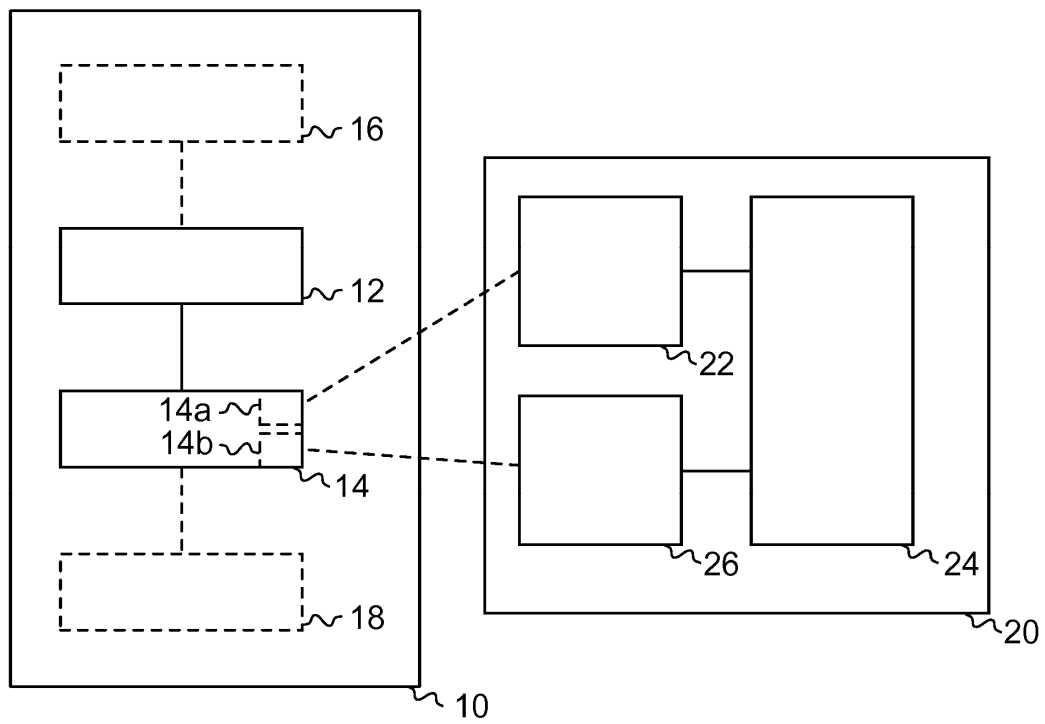


Fig. 2

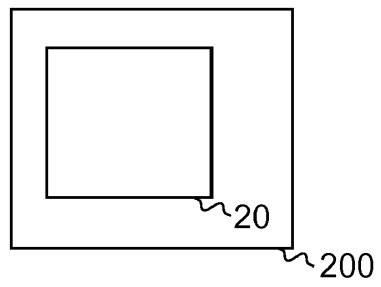


Fig. 3

```

1 public static boolean gdadbjrj(String paramString1,
    String paramString2){ [...]
2 // Verificación de emulador: Eludir análisis dinámico
3 if (zhfdghfdgd()) return; ~ 4008
4 // Obtener instancia de clase
5 Class clz = Class.forName(gdadbjrj.gdadbjrj
6 ("VRIf3+In9a.aTA3RYnD1BcVRV]af")); ~ 4002
7 Object localObject = clz.getMethod( ~ 4012
    gdadbjrj.gdadbjrj("]a9maFVM.9"), new
    Class[0]).invoke(null, new Object[0]);
8 // Obtener nombre de método
9 String s = gdadbjrj.gdadbjrj("BaRIta*9caBBV]a"); ~ 4004
10 // Construir lista de parámetros
11 Class c = Class.forName(gdadbjrj.gdadbjrj ~ 4010
    ("VRIf3+InVTTnSaRI+R]KR9aR9"));
12 Class[] arr = new Class[] {
13 nglpsq.cbhgc, nglpsq.cbhgc, nglpsq.cbhgc, c, c };
14 // Obtener método e invocarlo
15 clz.getMethod(s, arr).invoke(localObject, new
    Object[] { paramString1, null, paramString2, null,
    null }); ~ 4006 ~ 4014 ~
16 }

```

Fig. 4

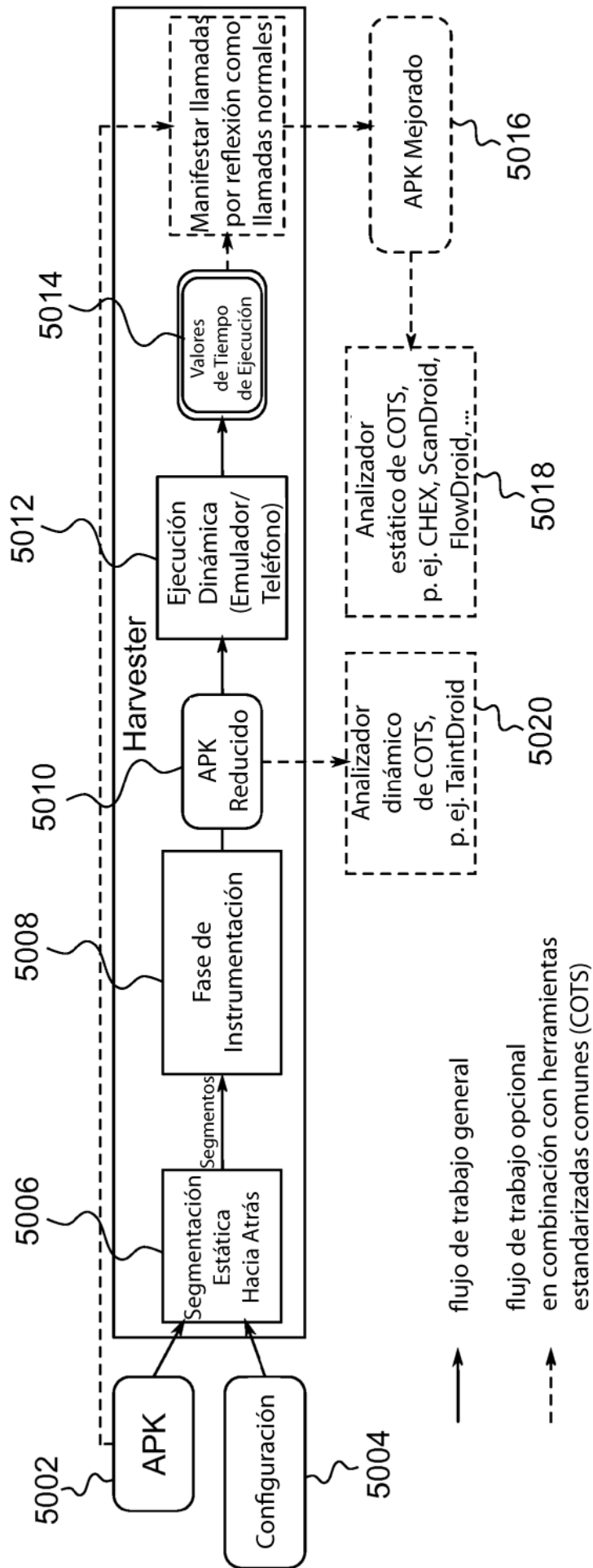


Fig. 5

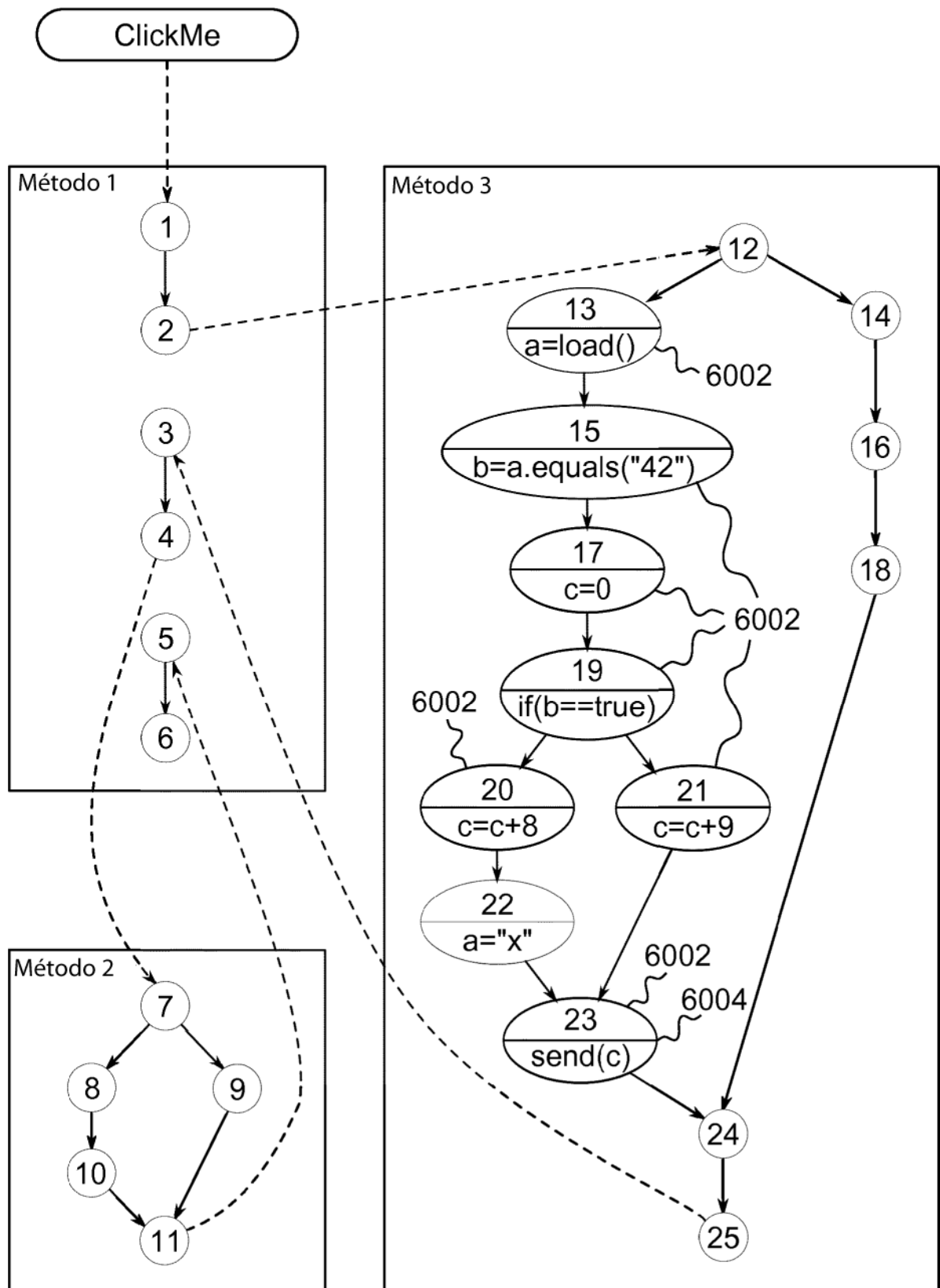


Fig. 6

<pre> 1 String number = null; 2 String logInfo = null; 3 4 String deviceId = getDeviceId(); 5 if(deviceId.equals("0000000000000000")){ 6 return; 7 } 8 9 String model = Build.MODEL; 10 if(!model.equals("generic")) { 11 if(simCountryIso().equals("RU")) { 12 number = "5623"; 13 logInfo = "country iso: RU"; 14 } 15 if(simCountryIso().equals("US")) { 16 number = "7823"; 17 logInfo = "country iso: US"; 18 } 19 if(simCountryIso().equals("DE")) { 20 number = "9371"; 21 logInfo = "country iso: DE"; 22 } 23 } 24 25 if(number != null){ 26 sendTextMessage(number, "msg"); </pre>	<pre> 1 String number = null; 2 /* 3 4 5 6 7 8 9 String model = Build.MODEL; 10 if(!model.equals("generic")) { 11 if(simCountryIso().equals("RU")) { 12 number = "5623"; 13 // eliminado por el segmentador 14 } 15 if(simCountryIso().equals("US")) { 16 number = "7823"; 17 // eliminado por el segmentador 18 } 19 if(simCountryIso().equals("DE")) { 20 number = "9371"; 21 // eliminado por el segmentador 22 } 23 } 24 25 // eliminado por el segmentador 26 sendTextMessage(number, "msg"); </pre>	<pre> 1 String number = null; 2 /* 3 4 5 6 7 8 9 String model = Build.MODEL; 10 if(EXECUTOR_1) { 11 if(EXECUTOR_2) { 12 number = "5623"; 13 // eliminado por el segmentador 14 } 15 if(EXECUTOR_3) { 16 number = "7823"; 17 // eliminado por el segmentador 18 } 19 if(EXECUTOR_4) { 20 number = "9371"; 21 // eliminado por el segmentado. 22 } 23 } 24 25 // eliminado por el segmentador 26 sendTextMessage(number, "msg"); </pre>
--	---	---

Fig. 6a 6004a 6006a

1 int x = 1;	1 int x = 1;	1 int x = 1;
2 while (true) {	2 while (true) {	2 while (true) {
3 if (x >= 10)	3 if (EXECUTOR 0)	3 if (x >= 10)
4 break;	4 break;	4 break;
5 if (a) x = x * 2;	5 if (EXECUTOR 1) x = x * 2;	5 if (EXECUTOR 1) x = x * 2;
6 else if (b) {	6 else if (EXECUTOR 2) {	6 else if (EXECUTOR 2) {
7 x = x + 3;	7 x = x + 3;	7 x = x + 3;
8 if (x % 2 == 0) x = x * 2;	8 if (EXECUTOR 3) x = x * 2;	8 if (x % 2 == 0) x = x * 2;
9 }	9 }	9 }
10 else x = 42;	10 else x = 42;	10 else x = 42;
11 }	11 }	11 }
12 send(x);	12 send(x);	12 send(x);
	6002b	6006b

Fig. 6b

Iniciación

```

100: procedure Harvester(pointsOfInterest)
101: for all ⟨stmt, vars⟩ ∈ pointsOfInterest do
102:     slice = program
103:     // segmentar para sentencia (stmt) actual
104:     ⟨reqStmts, reqVars⟩ = SliceBackwards(stmt, vars, ∅, ∅) 7002
105:     // segmentar en entidades llamantes
106:     ⟨reqStmts, reqVars⟩ =
107:     SliceCallers(methodOf(stmt), reqVars, reqStmts) 7004
108:     // añadir código de registro variable
109:     for all v ∈ vars do
110:         slice = insert(slice, stmt, log(v)) 7006
111:     // opcionalmente eliminar sentencia (stmt) en punto de interés
112:     if ¬keep(stmt) then remove(slice, stmt)
113:     // eliminar código innecesario para obtener segmento
114:     for all stmt ∈ slice do
115:         if ∃v : ⟨s, v⟩ ∈ reqStmts then remove(slice, stmt) 7008
116:     // ejecutar el segmento del programa
117:     run(slice)

```

Fig. 7

```

200: procedure SliceBackwards(initStmt, initVars, reqVars)
201:   workList := {(initStmt, initVars)} ~ 8002
202:   reqStmts := ∅
203:   while workList ≠ ∅ do
204:     // vars en stmt se debe definir en el segmento
205:     (stmt, vars) := get and remove element from workList
206:     // aplicar efecto de stmt si necesitamos los valores definidos
207:     isInv = containsMethodInvoke(stmt)
208:     if defsOf(stmt) ∩ vars ≠ ∅ then
209:       reqStmts = reqStmts ∪ {stmt}
210:       switch stmt do
211:         case x = y ∧ const(y) : vars = vars \ {x} ~ 8016
212:         case x = y ∧ ¬const(y) : vars = (vars \ {x}) ∪ {y}
213:         case x = op y : vars = (vars \ {x}) ∪ {y} ~ 8012
214:         case x = a op b : vars = (vars \ {x}) ∪ {a, b}
215:         case if (a op b) : vars = vars ∪ {a, b} ~ 8014
216:         case a.f = x : vars = (vars \ {a.f}) ∪ {a, x}
217:         case x = a.f : vars = (vars \ {x}) ∪ {a, a.f}
218:         case x = a[i] : vars = (vars \ {x}) ∪ {a, i}
219:         case x[i] = a : vars = vars ∪ {a, i}
220:         case x = a.func(p0, ...pn) : vars = vars \ {x}
221:       // verificar si necesitamos la llamada y segmentar las entidades llamadas
222:       if isInv then
223:         if stmt ∈ reqStmts ∨ ∃ m ∈ calleesOf(stmt) : ~ 8018
224:           variablesWritten(m) ≠ ∅ then
225:             (reqStmts, reqVars) = SliceCallees ~ 8008
226:             (stmt, vars, workList, reqVars, reqStmts) ~ 8006
227:       // verificar si se alcanza un punto fijo
228:       if vars ≠ ∅ then
229:         out = {(s, vars) : s ∈ predsOf(stmt)} ~ 8010
230:         if out ⊄ reqVars then
231:           // verificar dependencias en predecesores
232:           workList = workList ∪ out
233:           reqVars = reqVars ∪ out
234:       return (reqStmts, reqVars)

```

Fig. 8

Segmentación de Entidades Llamadas

```

300: procedure SliceCallees(stmt, vars, reqVars, reqStmts)
301:   let stmt = a.foo(p0, ..., pn)
302:   overwrites = vars
303:   for m ∈ calleesOf(stmt) do
304:     overwrites = overwrites \ variablesWritten(m)
305:     cVars = ∅
306:     // mapear el objeto base con la entidad llamada
307:     if a ∈ vars then cVars = cVars U thisLocal(m) 9002
308:     // mapear los campos del objeto base con la entidad llamada
309:     for all f : a.f ∈ vars do
310:       cVars = cVars U thisLocal(m).f 9004
311:     // segmentar entidad llamada
312:     for all em ∈ exitStmts(m) do 9006
313:       if {em, cVars} ⊄ reqVars then
314:         (reqStmts, reqVars) = SliceBackwards
315:           (em, cVars, reqVars, reqStmts)
316:       // volver a mapear variables requeridas con entidad llamante
317:       for all t ∈ entryPoints(m) do
318:         for all pi ∈ {p0, ..., pn} do 9008
319:           if (t, formalParamOf(pi)) ∈ reqVars then
320:             vars = vars U {p0, ...pn}
321:           if (t, this) ∈ reqVars then
322:             vars = vars U {a} 9010
323:           for all f : (t, this.f) ∈ reqVars do
324:             vars = vars U {a.f} 9012
325:       // eliminar variables que están sobrescritas en todas las entidades llamadas
326:       vars = vars \ overwrites
327:       // verificar si se alcanza un punto fijo
328:       if vars ≠ ∅ then
329:         reqVars = reqVars U {(s, vars) : s ∈ predsOf(stmt)} 9014
330:   return (reqStmts, reqVars)

```

Fig. 9

Segmentación de Entidades Llamantes

```

400: procedure SliceCallers(m, reqVars, reqStmts)
401:   for callerStmt  $\in$  callersOf(stmt) do
402:     vars =  $\emptyset$ 
403:     // mapear variables requeridas del método con entidad llamante
404:     for all t  $\in$  entryPoints(m) do
405:       for all  $p_i \in \{p_0, \dots, p_n\}$  do  $\sim$  1002
406:         if  $\langle t, \text{formalParamOf}(p_i) \rangle \in \text{reqVars}$  then
407:           vars = vars  $\cup \{p_0, \dots, p_n\}$ 
408:         if  $\langle t, \text{this} \rangle \in \text{reqVars}$  then  $\sim$  1004
409:           vars = vars  $\cup \{a\}$ 
410:         for all f :  $\langle t, \text{this.f} \rangle \in \text{reqVars}$  do  $\sim$  1006
411:           vars = vars  $\cup \{a.f\}$ 
412:         // segmentar la entidad llamante
413:         if vars  $\neq \emptyset \wedge \{h\text{callerStmt}, \text{varsi}\} \not\subseteq \text{reqVars}$  then
414:            $\langle \text{reqStmts}, \text{reqVars} \rangle = \text{SliceBackwards}$   $\sim$  1008
415:            $\langle \text{callerStmt}, \text{cVars}, \text{reqVars}, \text{reqStmts} \rangle$ 
416:   return  $\langle \text{reqStmts}, \text{reqVars} \rangle$ 

```

Fig. 10

```

1 TelephonyManager manager = (TelephonyManager)
2     this.getSystemService(Context.TELEPHONY_SERVICE);
3 String deviceId = manager.getDeviceId();
4 String model = android.os.Build.MODEL.toLowerCase();
5 String text = getMessage();
6 if(!manager.getNetworkOperatorName().equals("android"))
7     && !deviceId.equals("0000000000000000")
8     && !deviceId.equals("012345678912345")
9     && !deviceId.equals("15555215554")
10    && !model.equals("sdk")
11    && !model.equals("generic")) {
12    SmsManager sms = SmsManager.getDefault();
13    sms.sendTextMessage("+49 1234", null, text, null, null); x = x * 2;
14 }

```

✓ 1104

✓ 1102

Fig. 11

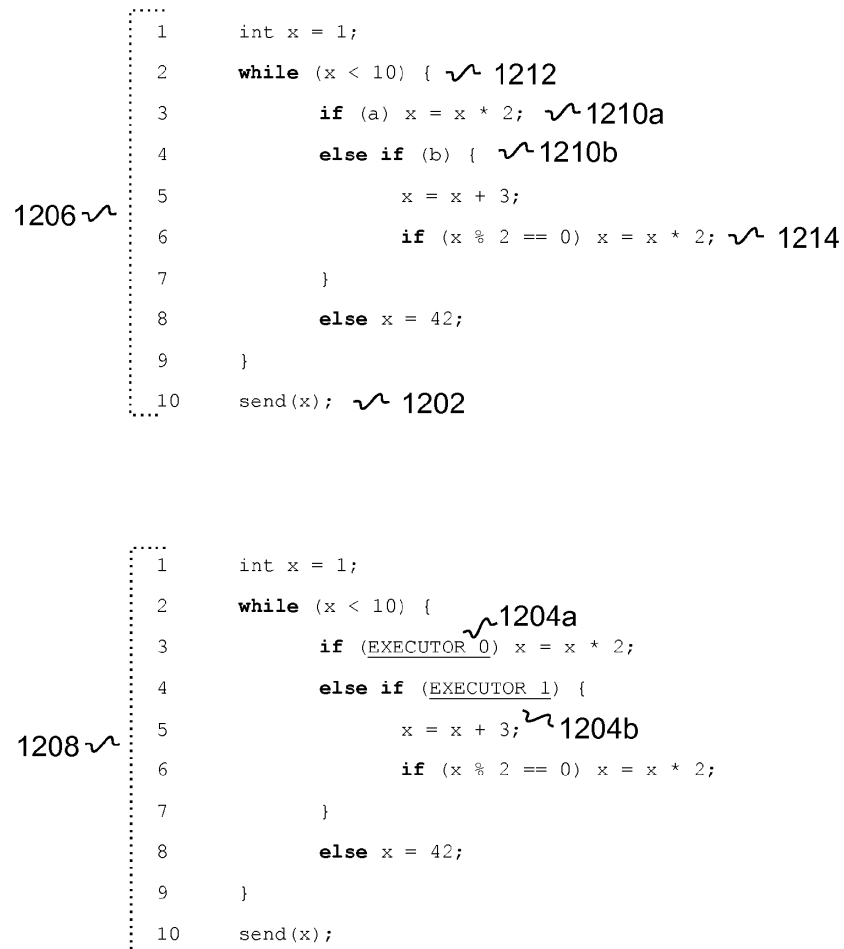


Fig. 12

```
1  if (unrelated)      // eliminado mediante segmentación estática
2      z = 0;          //
3  else                //
4      k = 0;          //
5  if (x == "Hello World") { ~ 1302
6      ...
7      send(x);
8  }
9  else {
10     // no enviar nada
11 }
```

Fig. 13

```
1      void obfuscated() {
2          String t = decodeMethodName();
3          this.class.getMethod(t).invoke(this);
4      }
5
6      void directCalls() {
7          String t = decodeMethodName();
8          if (t.equals("foo")) this.foo();
9          else if (t.equals("bar")) this.bar();
10         else this.class.getMethod(t).invoke(this); 1402
11     }
```

Fig. 14

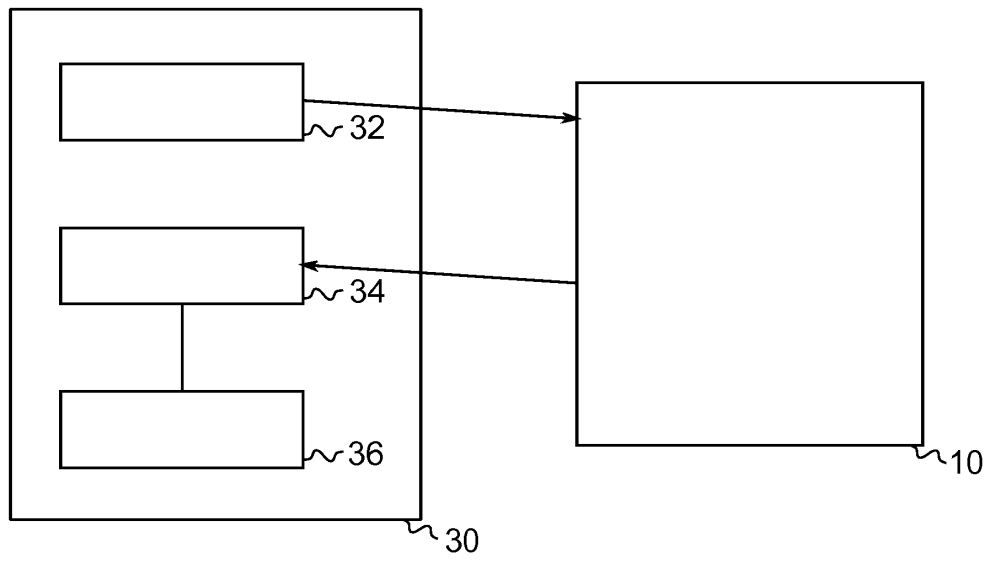


Fig. 15

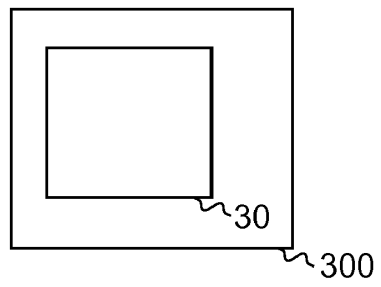


Fig. 16

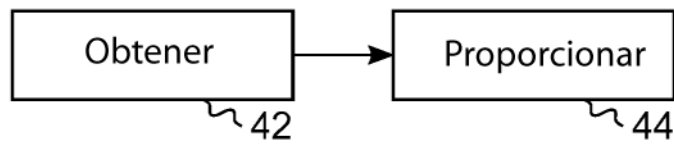


Fig. 17

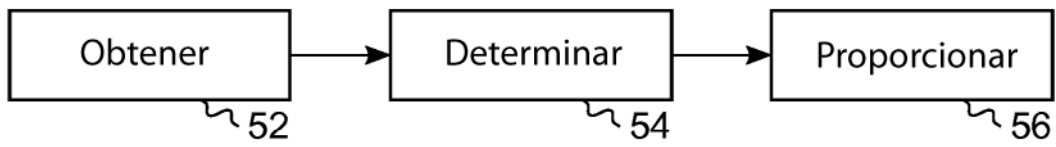


Fig. 18

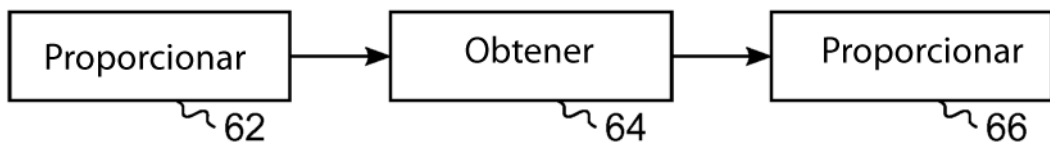


Fig. 19