(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 988 317**

(51) Int. Cl.:

G06F 21/54 (2013.01)
G06F 21/52 (2013.01)
G06F 21/55 (2013.01)
G11C 29/52 (2006.01)
G06F 12/14 (2006.01)
G06F 12/02 (2006.01)
G06F 9/445 (2008.01)
G06F 9/448 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **02.10.2019 PCT/IL2019/051075**

(87) Fecha y número de publicación internacional: **23.04.2020 WO20079676**

(96) Fecha de presentación y número de la solicitud europea: **02.10.2019 E 19873518 (5)**

(97) Fecha y número de publicación de la concesión europea: **24.07.2024 EP 3867784**

(54) Título: **Aplicación de medidas de mitigación de seguridad para la explotación de corrupción de pila en archivos de código intermedio**

(30) Prioridad:

18.10.2018 US 201862747150 P

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
20.11.2024

(73) Titular/es:

STERNUM LTD. (100.0%)
94 Yigal Alon Street,Tower 2 Floor 9
6744317 Tel Aviv-Yafo, IL

(72) Inventor/es:

TSHOUVA, NATALI;
GRANOT, LIAN;
FARBER, ARIK y
GRANOT, TAL

(74) Agente/Representante:

LINAGE GONZÁLEZ, Rafael

DESCRIPCIÓN

Aplicación de medidas de mitigación de seguridad para la explotación de corrupción de pila en archivos de código intermedio

Antecedentes

La presente invención, en algunas realizaciones de la misma, se relaciona con la aplicación de protección contra la explotación de corrupción de pila y, más específicamente, pero no exclusivamente, con el ajuste de archivos de código intermedio para aplicar protección contra la explotación de corrupción de pila.

En el entorno de los módems computarizados en constante avance y evolución, las amenazas cibernéticas se han convertido en una preocupación importante. Las partes maliciosas pueden lanzar ataques cibernéticos contra múltiples plataformas, aplicaciones y/o servicios en un intento de obtener control sobre ellos para una pluralidad de metas y/u objetivos que van desde el hackeo inofensivo hasta la explotación para obtener ganancias financieras e incluso la interferencia maliciosa en sistemas críticos.

Estas preocupaciones pueden intensificarse drásticamente con el rápido despliegue de una gran cantidad de dispositivos, usualmente pequeños dispositivos integrados de gama baja utilizados para respaldar la automatización en una pluralidad de campos, áreas y mercados, por ejemplo, vehículos autónomos, ciudades inteligentes, agricultura, atención médica y procedimientos médicos y/o similares. Estos dispositivos integrados de gama baja usualmente pueden tener recursos limitados que pueden ser insuficientes para aplicar medidas de mitigación sofisticadas para contrarrestar dichas amenazas cibernéticas.

Sin embargo, estos dispositivos integrados con recursos limitados pueden estar involucrados en aplicaciones críticas, servicios y/o similares y pueden servir opcionalmente como puntos de acceso a plataformas y sistemas de nivel superior. Por lo tanto, los dispositivos integrados pueden presentar una importante brecha de seguridad que puede ser explotada por partes maliciosas. El documento “G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries (*G-Libre: Venciendo la Programación Orientada al Retorno mediante Binarios sin Dispositivos*)” de K. Onarlioglu et al. en “Proceedings of the 26th Annual Computer Security Applications Conference (*Actas de la 26ª Conferencia Anual sobre Aplicaciones de Seguridad Informática*)”, 2010, divulga una solución de enfoque basado en compilador contra cualquier tipo de programación orientada al retorno.

Sumario

La invención se define por las reivindicaciones adjuntas. De acuerdo con un primer aspecto de la presente invención, se proporciona un procedimiento implementado por ordenador para generar archivos de código intermedio compilados ajustados para aplicar protección de dirección de retorno, que comprende:

- Recibir uno o más archivos de código intermedio generados por un compilador, comprendiendo el o los archivos de código intermedio una pluralidad de rutinas.
- Ajustar uno o más de los archivos de código intermedio antes de la generación de un respectivo archivo ejecutable para ser ejecutado por uno o más procesadores. Comprendiendo el ajuste:
    - ▪ Analizar una tabla de símbolos de uno o más de los archivos de código intermedio para identificar cada una de la pluralidad de rutinas.
    - ▪ Ajustar una o más de la pluralidad de rutinas para reemplazar una instrucción de inserción en pila de dirección de retorno detectada en las respectivas rutinas con un respectivo segmento de código de prólogo y para reemplazar cada instrucción de extracción de pila de dirección de retorno detectada en las respectivas rutinas con un respectivo segmento de código de epílogo. El respectivo segmento de código de prólogo está configurado para alterar la pila después de que la dirección de retorno se inserta en la pila y el respectivo segmento de código de epílogo está configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes de ramificarse a la dirección de retorno recuperada de la pila.
- Emitir los archivos de código intermedio ajustados,

en el que, en tiempo de ejecución, en caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que uno o más de los procesadores inicien una o más acciones predefinidas.

De acuerdo con un segundo aspecto de la presente invención, se proporciona un sistema para generar archivos de código intermedio compilados ajustados para aplicar protección de dirección de retorno, que comprende un almacén de programas que almacena un código y uno o más procesadores acoplados al almacén de programas para ejecutar el código almacenado, comprendiendo el código:

- Instrucciones de código para recibir uno o más archivos de código intermedio generados por un compilador, comprendiendo el o los archivos de código intermedio una pluralidad de rutinas;
- Instrucciones de código para ajustar uno o más de los archivos de código intermedio antes de la generación de un respectivo archivo ejecutable para ser ejecutado por uno o más procesadores, comprendiendo el ajuste:

    ▪ Analizar una tabla de símbolos de uno o más de los archivos de código intermedio para identificar cada una de la pluralidad de rutinas.
    ▪ Ajustar una o más de la pluralidad de rutinas para reemplazar una instrucción de inserción en pila de dirección detectada en las respectivas rutinas con un respectivo segmento de código de prólogo y para reemplazar cada instrucción de extracción de pila de dirección detectada en las respectivas rutinas con un respectivo segmento de código de epílogo. El respectivo segmento de código de prólogo está configurado para alterar la pila después de que la dirección de retorno se inserta en la pila y el respectivo segmento de código de epílogo está configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes de ramificarse a la dirección de retorno recuperada de la pila.

- Instrucciones de código para emitir el o los archivos de código intermedio ajustados;

en el que, en tiempo de ejecución, en caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que uno o más de los procesadores inicien una o más acciones predefinidas.

De acuerdo con un tercer aspecto de la presente invención, se proporciona un archivo ejecutable de programa informático generado a partir de archivos de código intermedio ajustados para aplicar protección de dirección de retorno, que comprende un medio de almacenamiento legible por ordenador no transitorio y una pluralidad de instrucciones de programa de una o más rutinas ajustadas de un archivo ejecutable generado para su ejecución por uno o más procesadores a partir de uno o más archivos de código intermedio ajustados para soportar la protección de dirección de retorno. En cada una de las rutinas ajustadas, una instrucción de inserción en pila de dirección se reemplaza con un respectivo segmento de código de prólogo y cada instrucción de extracción de pila de dirección se reemplaza con un respectivo segmento de código de epílogo. El respectivo segmento de código de prólogo está configurado para alterar la pila después de que la dirección de retorno se inserta en la pila y el respectivo segmento de código de epílogo está configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes de ramificarse a la dirección de retorno recuperada de la pila. En caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que uno o más de los procesadores inicien una o más acciones predefinidas. En el que la pluralidad de instrucciones de programa son ejecutadas por uno o más de los procesadores desde el medio de almacenamiento legible por ordenador no transitorio.

De acuerdo con un cuarto aspecto de la presente invención, se proporciona un procedimiento implementado por ordenador para generar archivos de código intermedio compilados ajustados para evitar la explotación de la programación orientada al retorno, que comprende:

- Recibir uno o más archivos de código intermedio generados por un compilador, comprendiendo el o los archivos de código intermedio una pluralidad de rutinas.
- Ajustar uno o más de los archivos de código intermedio antes de la generación de un respectivo archivo ejecutable para ser ejecutado por uno o más procesadores. Comprendiendo el ajuste:

    ▪ Analizar una tabla de símbolos de uno o más de los archivos de código intermedio para identificar una dirección de inicio de cada una de la pluralidad de rutinas.
    ▪ Analizar cada una de la pluralidad de rutinas para identificar una o más instrucciones de ramificación indirecta en las respectivas rutinas.
    ▪ Reemplazar cada instrucción de ramificación indirecta detectada en una o más de la pluralidad de rutinas con una invocación de un respectivo segmento de código de verificación configurado para verificar, antes de ejecutar la respectiva operación de ramificación indirecta, que la respectiva instrucción de ramificación indirecta apunta al inicio de una de la pluralidad de rutinas.

- Emitir los archivos de código intermedio ajustados.

en el que, en tiempo de ejecución, en caso de que la instrucción de ramificación indirecta no apunte al inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que uno o más de los procesadores inicien una o más acciones predefinidas.

De acuerdo con un quinto aspecto de la presente invención, se proporciona un sistema para generar archivos de código intermedio compilados ajustados para evitar la explotación de la programación orientada al retorno, que

comprende un almacén de programas que almacena un código y uno o más procesadores acoplados al almacén de programas para ejecutar el código almacenado, comprendiendo el código:

- Instrucciones de código para recibir uno o más archivos de código intermedio generados por un compilador, comprendiendo el o los archivos de código intermedio una pluralidad de rutinas.
- Instrucciones de código para ajustar uno o más de los archivos de código intermedio antes de la generación de un respectivo archivo ejecutable para ser ejecutado por uno o más procesadores, comprendiendo el ajuste:

    ▪ Analizar una tabla de símbolos de uno o más de los archivos de código intermedio para identificar una dirección de inicio de cada una de la pluralidad de rutinas.
    ▪ Analizar cada una de la pluralidad de rutinas para identificar una o más instrucciones de ramificación indirecta en las respectivas rutinas.
    ▪ Reemplazar cada instrucción de ramificación indirecta detectada en cada una de la pluralidad de rutinas por una invocación de un respectivo segmento de código de verificación configurado para verificar, antes de ejecutar la respectiva operación de ramificación indirecta, que la respectiva instrucción de ramificación indirecta apunta al inicio de una de la pluralidad de rutinas.

- Instrucciones de código para generar el o los archivos de código intermedio ajustados;

en el que, en tiempo de ejecución, en caso de que la instrucción de ramificación indirecta no apunte al inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que uno o más de los procesadores inicien una o más acciones predefinidas.

De acuerdo con un sexto aspecto de la presente invención, se proporciona un archivo ejecutable de programa informático generado a partir de un archivo de código intermedio ajustado para evitar la explotación de la programación orientada al retorno, que comprende un medio de almacenamiento legible por ordenador no transitorio y una pluralidad de instrucciones de programa de una o más rutinas ajustadas de una pluralidad de rutinas de un archivo ejecutable generado para su ejecución por uno o más procesadores a partir de uno o más archivos de código intermedio ajustados para soportar la protección de la dirección de retorno. En cada una de las rutinas ajustadas, cada instrucción de ramificación indirecta se reemplaza con una invocación de un respectivo segmento de código de verificación configurado para verificar que la respectiva instrucción de ramificación indirecta apunta al inicio de una de la pluralidad de rutinas. En caso de que la instrucción de ramificación indirecta no apunte al inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que uno o más de los procesadores inicien una o más acciones predefinidas. En el que la pluralidad de instrucciones de programa son ejecutadas por uno o más de los procesadores desde el medio de almacenamiento legible por ordenador no transitorio.

En una forma de implementación adicional del primer, segundo, tercer, cuarto, quinto y/o sexto aspecto, cada archivo de código intermedio es un miembro de un grupo que consiste en: un archivo de objeto, un archivo de almacenamiento y un archivo binario.

En una forma de implementación adicional del primer, segundo, tercer, cuarto, quinto y/o sexto aspecto, cada una de la pluralidad de rutinas es un miembro de un grupo que consiste en: una rutina, una subrutina y una función.

En una forma de implementación adicional del primer, segundo, tercer, cuarto, quinto y/o sexto aspecto, la acción predefinida son miembros de un grupo que consiste en: bloquear la ejecución de uno o más procesadores, detener la ejecución de uno o más procesadores, hacer que uno o más procesadores se ramifiquen a una dirección predefinida, evitar que uno o más procesadores ejecuten una o más instrucciones de código potencialmente malicioso y generar una indicación de una alteración de pila inválida.

En una forma de implementación adicional del primer, segundo, tercer, cuarto, quinto y/o sexto aspecto, uno o más de los archivos de código intermedio se ajustan para modificar uno o más de: una instrucción y un elemento de datos afectado por la adición de los segmentos de código agregados.

En una forma de implementación adicional del primer, segundo, tercer, cuarto, quinto y/o sexto aspecto, uno o más de los archivos de código intermedio se modifican para actualizar su tabla de símbolos para reflejar los segmentos de código agregados y un aumento en el tamaño de las rutinas ajustadas.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, en caso de que la instrucción de inserción en pila de dirección esté asociada con la inserción de uno o más elementos de datos a la pila, el respectivo segmento de código de prólogo está configurado para insertar el elemento o elementos de datos a la pila y el respectivo segmento de código de epílogo está configurado para extraer el elemento o elementos de datos de la pila.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, la alteración de la pila se basa en alterar la dirección de retorno de una manera reversible mediante:

- Configurar el respectivo segmento de código de prólogo para leer la dirección de retorno insertada en la pila, alterar la dirección de retorno de una manera reversible e insertar la dirección de retorno alterada de nuevo en la pila.
- Configurar el respectivo segmento de código de epílogo para leer la dirección de retorno alterada de la dirección de pila, recuperar la dirección de retorno de la dirección de retorno alterada invirtiendo la operación realizada por el respectivo segmento de código de prólogo e insertar la dirección de retorno recuperada de nuevo en la pila.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, la alteración de la dirección de retorno se basa en una operación XOR de la dirección de retorno con un valor seleccionado aleatoriamente de tal manera que el respectivo segmento de código de prólogo está configurado para realizar la operación XOR en la dirección de retorno con el valor seleccionado aleatoriamente y el respectivo segmento de código de epílogo está configurado para realizar la operación XOR en la dirección de retorno alterada con el mismo valor seleccionado aleatoriamente.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, la alteración de la pila se basa en insertar un marcador en la pila en una ubicación adyacente a la ubicación de dirección de retorno insertada mediante:

- Configurar el respectivo segmento de código de prólogo para insertar un marcador de valor constante en la ubicación adyacente en la pila.
- Configurar el respectivo segmento de código de epílogo para verificar el marcador de valor constante y eliminar el marcador de valor constante de la pila.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, el marcador de valor constante se selecciona aleatoriamente durante cada evento de inicio de uno o más de los procesadores.

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, la alteración de la pila se basa en insertar un marcador en la pila en una ubicación adyacente a la ubicación de dirección de retorno insertada mediante:

En una forma de implementación adicional del primer, segundo y/o tercer aspecto, el respectivo segmento de código de prólogo, los respectivos segmentos de código de epílogo y/o el segmento de código de verificación se agregan a la(s) rutina(s) sustituyendo la instrucción de inserción en pila de dirección y cada una de las instrucciones de extracción de pila de dirección por una función de ramificación de trampolín configurada para invocar el respectivo segmento de código agregado durante las operaciones de inserción y extracción de dirección.

En una forma de implementación adicional del cuarto, quinto y/o sexto aspecto, la verificación de la instrucción de ramificación indirecta que apunta al inicio de una de la pluralidad de rutinas se basa en la verificación de un código único que precede a cada una de la pluralidad de rutinas en uno o más de los archivos de código intermedio, la verificación de código único se aplica mediante:

- Agregar un código único a uno o más de los archivos de código intermedio en una dirección que precede a la dirección del inicio de cada una de la pluralidad de rutinas.
- Configurar el respectivo segmento de código de verificación para verificar que la dirección apuntada por la función de ramificación indirecta esté precedida por el código único.

En una forma de implementación adicional del cuarto, quinto y/o sexto aspecto, la verificación de la instrucción de ramificación indirecta que apunta al inicio de una de la pluralidad de rutinas se basa en un conjunto de datos que mapea el inicio de cada una de la pluralidad de rutinas, la verificación del mapeo del conjunto de datos se aplica mediante:

- Construir el conjunto de datos para mapear la dirección de inicio de cada una de la pluralidad de rutinas.
- Configurar el respectivo segmento de código de verificación para verificar que la dirección apuntada por la función de ramificación indirecta coincida con la dirección de inicio de una de la pluralidad de rutinas mapeadas en el conjunto de datos.

En una forma de implementación adicional del cuarto, quinto y/o sexto aspecto, la(s) rutina(s) se ajusta(n) reemplazando la instrucción de ramificación indirecta con una instrucción de ramificación directa que apunta al respectivo segmento de código de verificación.

En una forma de implementación opcional del cuarto, quinto y/o sexto aspecto, la(s) rutina(s) se ajusta(n) para incluir una instrucción de inserción en pila antes de la instrucción de ramificación directa para insertar en pila una dirección apuntada por la instrucción de ramificación indirecta. La dirección insertada es utilizada por el segmento de código de verificación para la verificación.

En una forma de implementación adicional del cuarto, quinto y/o sexto aspecto, el segmento de código de verificación se agrega a la(s) rutina(s) reemplazando la instrucción de ramificación indirecta con una función de ramificación de trampolín configurada para invocar el respectivo segmento de código de verificación agregado antes de ejecutar la instrucción de ramificación indirecta.

Otros sistemas, procedimientos, características y ventajas de la presente divulgación serán o se harán evidentes para una persona con experiencia en la técnica tras examinar los siguientes dibujos y la descripción detallada. Se pretende que todos esos sistemas, procedimientos, características y ventajas adicionales se incluyan en esta descripción, estén dentro del alcance de la presente divulgación y estén protegidos por las reivindicaciones adjuntas.

A menos que se defina lo contrario, todos los términos técnicos y/o científicos utilizados en la presente memoria descriptiva tienen el mismo significado que entiende comúnmente una persona con experiencia ordinaria en la técnica a la que pertenece la invención. Aunque se pueden utilizar procedimientos y materiales similares o equivalentes a los descritos en la presente memoria descriptiva en la práctica o prueba de realizaciones de la invención, a continuación, se describen procedimientos y/o materiales ejemplares. En caso de conflicto, prevalecerá la memoria descriptiva de la patente, incluidas las definiciones. Además, los materiales, procedimientos y ejemplos son solo ilustrativos y no pretenden ser necesariamente limitantes.

La implementación del procedimiento y/o sistema de realizaciones de la invención puede implicar la realización o finalización de tareas seleccionadas de forma manual, automática o una combinación de las mismas. Además, de acuerdo con la instrumentación y el equipo reales de las realizaciones del procedimiento y/o sistema de la invención, varias tareas seleccionadas podrían implementarse mediante hardware, software o firmware o una combinación de los mismos utilizando un sistema operativo.

Por ejemplo, el hardware para realizar tareas seleccionadas de acuerdo con realizaciones de la invención podría implementarse como un chip o un circuito. Como software, las tareas seleccionadas de acuerdo con realizaciones de la invención podrían implementarse como una pluralidad de instrucciones de software que son ejecutadas mediante un ordenador utilizando cualquier sistema operativo adecuado. En una realización ejemplar de la invención, una o más tareas de acuerdo con realizaciones ejemplares del procedimiento y/o sistema como se describe en la presente memoria descriptiva son realizadas por un procesador de datos, tal como una plataforma informática para ejecutar una pluralidad de instrucciones. Opcionalmente, el procesador de datos incluye una memoria volátil para almacenar instrucciones y/o datos y/o un almacenamiento no volátil, por ejemplo, un disco duro magnético y/o un medio extraíble, para almacenar instrucciones y/o datos. Opcionalmente, también se proporciona una conexión de red. También se proporcionan opcionalmente una pantalla y/o un dispositivo de entrada de usuario, como un teclado o un ratón.

Breve descripción de las diversas vistas de los dibujos

En la presente memoria descriptiva se describen algunas realizaciones de la invención, únicamente a modo de ejemplo, con referencia a los dibujos adjuntos. Con referencia específica ahora a los dibujos en detalle, se destaca que los detalles mostrados son a modo de ejemplo y con fines de discusión ilustrativa de realizaciones de la invención. En este sentido, la descripción tomada con los dibujos hace evidente para los expertos en la técnica cómo se pueden llevar a la práctica las realizaciones de la invención.

En los dibujos:

La Figura 1 es un diagrama de flujo de un proceso ejemplar de ajuste de archivos de código intermedio para aplicar protección de dirección de retorno, de acuerdo con algunas realizaciones de la presente invención;
La Figura 2 es una ilustración esquemática de un sistema ejemplar para ajustar archivos de código intermedio para aplicar protección contra explotación de corrupción de memoria, de acuerdo con algunas realizaciones de la presente invención; y
La Figura 3 es un diagrama de flujo de un proceso ejemplar de ajuste de archivos de código intermedio para aplicar protección contra explotación de programación orientada al retorno, de acuerdo con algunas realizaciones de la presente invención.

Descripción detallada

La presente invención, en algunas realizaciones de la misma, se relaciona con la aplicación de protección contra explotación de corrupción de pila, y, más específicamente, pero no exclusivamente, con el ajuste de archivos de

código intermedio para aplicar protección contra explotación de corrupción de pila.

De acuerdo con algunas realizaciones de la presente invención, se proporcionan procedimientos, sistemas y productos de programas informáticos para ajustar uno o más archivos de código intermedio, por ejemplo, un archivo de objeto, un archivo binario, un archivo de biblioteca, un archivo de almacenamiento y/o similares generados a partir de uno o más archivos de código fuente por uno o más compiladores para incluir código adicional configurado para aplicar Protección de Dirección de Retorno (RAP) en el(los) archivo(s) de código intermedio. El(los) archivo(s) de código intermedio comprenden una pluralidad de rutinas, subrutinas, funciones y/o similares, denominados colectivamente rutina en lo sucesivo, se ajustan antes de ser utilizados para la generación, por ejemplo, construcción, vinculación y/o similares de uno o más archivos ejecutables. El(los) archivo(s) ejecutable(s) puede(n) ser ejecutado(s) por uno o más procesadores de uno o más dispositivos, sistemas y/o plataformas, denominados colectivamente dispositivo en lo sucesivo.

Un malware (código malicioso), por ejemplo, un virus informático, un gusano, un caballo de Troya, un ransomware, un spyware, un adware, un scareware y/o similares, puede ser utilizado por una o más partes potencialmente maliciosas para obtener el control sobre el flujo de control (ejecución) del procesador o procesadores. El malware puede aplicar una o más vulnerabilidades de seguridad, por ejemplo, sobreflujo de búfer, desbordamiento de búfer y/o similares para manipular la pila (implementada en un recurso de memoria volátil) del procesador o procesadores en un intento de obtener el control sobre el procesador o procesadores utilizando la dirección de retorno al regresar de una o más de las rutinas ejecutadas por el procesador o procesadores. Al provocar un sobreflujo de búfer, es posible que el malware coloque código malicioso en áreas que se sabe que contienen código ejecutable o sobrescriba selectivamente datos pertenecientes al estado del programa, lo que provoca un comportamiento que no fue provisto por el programador original.

Para manipular la pila en tiempo de ejecución, el malware puede aprovechar las instrucciones de inserción en pila y/o extracción de pila presentes en el código para operaciones de ramificación en las que las instrucciones de inserción/extracción en/de pila se utilizan para operaciones de almacenamiento y recuperación de direcciones de retorno aplicadas para facilitar el retorno a una rutina de origen (llamada) desde la que se llamó a la rutina que se está ejecutando actualmente. Por lo tanto, las instrucciones de inserción/extracción en/de pila son susceptibles de explotación por parte del malware. En otro ejemplo, el malware puede aprovechar las instrucciones de inserción/extracción en/de pila presentes en el código para punteros a elementos de datos, en particular elementos de datos globales, por ejemplo, una variable, una estructura, una matriz y/o similares. En tal caso, el malware puede intentar reemplazar y/o desbordar uno o más de los elementos de datos apuntados con código malicioso a fin de obtener el control sobre el flujo de control del procesador o procesadores y/o hacer que el procesador o procesadores realice(n) operaciones no deseadas.

A fin de aplicar la RAP en el(los) archivo(s) de código intermedio, una o más de las rutinas en el(los) archivo(s) de código intermedio se pueden ajustar para incluir lógica de RAP (código) adaptada para identificar la manipulación de la pila en tiempo de ejecución y evitar que el malware tome control del flujo de control del procesador o procesadores al manipular la pila. Cada archivo de código intermedio puede analizarse primero para identificar la pluralidad de rutinas. Cada una de las rutinas puede analizarse además para identificar las instrucciones de inserción en pila de dirección de retorno y las correspondientes instrucciones de extracción de pila de dirección de retorno. Las instrucciones de inserción en pila de dirección de retorno identificadas y las correspondientes instrucciones de extracción de pila de dirección de retorno pueden reemplazarse con segmentos de código configurados para identificar la(s) manipulación(es) realizadas en la pila por el malware.

Cada instrucción de inserción en pila de dirección de retorno puede reemplazarse con un segmento de código de prólogo configurado para alterar la pila de una determinada manera después de que la dirección de retorno se inserte en la pila. De manera complementaria, cada instrucción de extracción de pila de dirección de retorno puede reemplazarse con un segmento de código de epílogo configurado para validar la alteración de la pila realizada por el segmento de código de prólogo correspondiente antes de ramificarse a la dirección de retorno recuperada de la pila.

Durante el tiempo de ejecución, en caso de que el segmento de código de epílogo no pueda verificar la alteración de la pila realizada por el segmento de código de prólogo correspondiente, el segmento de código de epílogo puede iniciar una o más acciones predefinidas dirigidas a evitar que el malware tome el control. Las acciones predefinidas pueden incluir, por ejemplo, bloquear la ejecución del procesador o procesadores, detener la ejecución del procesador o procesadores, hacer que el procesador o procesadores se ramifiquen a una dirección predefinida, evitar que el procesador o procesadores ejecuten al menos una instrucción de código potencialmente malicioso dirigida por una pila corrupta, generar una o más indicaciones y/o alertas de alteración de pila inválida y/o similares. Las acciones predefinidas pueden seleccionarse de acuerdo con uno o más parámetros del flujo de control (ejecución), por ejemplo, la arquitectura del procesador, una gravedad y/o criticidad de cada rutina, un parámetro definido por el usuario y/o similares.

Los segmentos de código agregados comprenden usualmente una lógica muy simple implementada por un

número limitado y significativamente pequeño de instrucciones, por lo que tienen una huella muy pequeña que requiere recursos de almacenamiento y/o recursos informáticos insignificantes.

De acuerdo con algunas realizaciones de la presente invención, se proporcionan procedimientos, sistemas y productos de programas informáticos para ajustar uno o más de los archivos de código intermedio para incluir código agregado configurado para evitar la Programación Orientada al Retorno (ROP).

En muchos dispositivos, sistemas, plataformas y/o entornos informáticos, es posible que las partes maliciosas no puedan ejecutar código desde el montón y/o desde la pila de los procesadores y que no puedan inyectar (agregar, insertar, etc.) malware en el entorno de código ejecutable. En tales casos, las partes maliciosas pueden aprovechar uno o más elementos de código (también denominados *gadgets*), por ejemplo, una función, un segmento de código, una fracción de código y/o similares que ya estén disponibles en el archivo ejecutable. Las partes maliciosas pueden utilizar dichos gadgets para desviar los procesadores que ejecutan el archivo ejecutable de su ruta de ejecución normal a una ruta de ejecución alternativa que puede presentar una o más amenazas cibernéticas potenciales.

En tales dispositivos, sistemas, plataformas y/o entornos informáticos, el flujo de ejecución siempre puede ramificarse para dirigirse a ubicaciones que son las direcciones iniciales de las rutinas. Sin embargo, para lanzar, ejecutar y/o iniciar los gadgets, la parte maliciosa puede aplicar ROP para explotar una o más operaciones de ramificación de rutina manipulando la pila de llamadas utilizada por la operación de ramificación para invocar el(los) gadget(s) en lugar de una rutina válida. En particular, la parte maliciosa puede explotar operaciones de ramificación indirecta donde la dirección de ramificación se recupera de un recurso de memoria volátil (por ejemplo, pila, montón, registro, etc.). Las operaciones de ramificación directa, en las que la dirección de ramificación está codificada, pueden no estar expuestas a la explotación de ROP ya que la dirección de ramificación no se recupera de un recurso de memoria volátil. Por lo tanto, la lógica y las medidas de prevención de explotación de ROP descritas en la presente memoria descriptiva están dirigidas a ramificaciones indirectas en lugar de a ramificaciones directas.

Con el fin de aplicar la prevención de explotación de ROP en el(los) archivo(s) de código intermedio, una o más de las rutinas en el(los) archivo(s) de código intermedio pueden ajustarse para incluir una lógica de prevención de explotación de ROP configurada para evitar la ramificación a ubicaciones de direcciones que no son el inicio de rutinas válidas, impidiendo así que la(las) parte(s) maliciosa(s) utilicen el o los gadgets que usualmente se encuentran dentro de las rutinas para obtener el control sobre el flujo de ejecución de los procesadores.

Cada archivo de código intermedio puede analizarse para identificar la pluralidad de rutinas. Cada una de las rutinas se analiza además para identificar instrucciones de ramificación indirecta. Cada una de las instrucciones de ramificación indirecta identificadas se puede ajustar para invocar un segmento de código de verificación configurado para verificar que la dirección de ramificación indicada por la respectiva instrucción de ramificación indirecta es una dirección de inicio de una de la pluralidad de rutinas.

Durante el tiempo de ejecución, en caso de que el segmento de código de verificación determine que la dirección de ramificación indicada por una o más de las instrucciones de ramificación indirecta no indica una dirección de inicio de una de la pluralidad de rutinas, el segmento de código de verificación puede iniciar una o más acciones predefinidas dirigidas a impedir que la(s) parte(s) maliciosa(s) obtengan el control sobre el flujo de ejecución del(de los) procesador(es) manipulando la instrucción de ramificación indirecta para que indique una dirección alternativa que puede ser la dirección del(de los) gadget(s).

La aplicación de las mitigaciones de seguridad de explotación de RAP y ROP mediante el ajuste del(de los) archivo(s) de código intermedio en la fase posterior a la compilación puede presentar ventajas y beneficios significativos en comparación con los procedimientos y sistemas existentes para incorporar medidas de explotación de RAP y ROP en el código.

En primer lugar, las medidas de mitigación de explotación de RAP y ROP se aplican en los archivos de código intermedio en comparación con los procedimientos existentes que pueden ser capaces de aplicar dichas medidas de mitigación en los archivos de código fuente. Por lo tanto, a diferencia de los procedimientos existentes, las medidas de mitigación de explotación de RAP y ROP pueden aplicarse a archivos de código intermedio para los que no está disponible el código fuente, por ejemplo, archivos de código intermedio de terceros, archivos de biblioteca y/o similares.

Además, algunos de los procedimientos existentes pueden requerir y/o depender de capacidades y funcionalidades específicas de plataforma, hardware y/o software, para aplicar la mitigación de explotación de RAP y/o ROP, por ejemplo, aislamiento de montón, invocación de tareas, separación de usuario y núcleo y/o similares. Por el contrario, la aplicación de las medidas de mitigación de explotación de RAP y ROP a través del ajuste de la(s) rutina(s) en el(los) archivo(s) de código intermedio es independiente de dichos requisitos y/o prerrequisitos y, por lo tanto, no tiene en cuenta y es agnóstica respecto del sistema operativo, las características

del hardware, las particularidades de la arquitectura y/o similares. Como tal, las medidas de mitigación de explotación de RAP y ROP pueden aplicarse prácticamente a cualquier plataforma, arquitectura, entorno de ejecución, sistema operativo y/o similares.

Además, algunos de los procedimientos existentes pueden requerir recursos del sistema operativo para aplicar las medidas de mitigación de explotación de RAP y ROP, por ejemplo, módulos de código cargables dinámicamente, llamadas de sistema, controladores de interrupciones y/o similares. Dichos módulos de código cargables dinámicamente pueden no estar disponibles para una pluralidad de dispositivos, sistemas y/o plataformas informáticas, en particular dispositivos de gama baja y/o recursos limitados, por ejemplo, dispositivos integrados, dispositivos IoT, dispositivos médicos y/o similares. Por el contrario, la aplicación de las medidas de mitigación de explotación de RAP y ROP mediante el ajuste de la(s) rutina(s) en el(los) archivo(s) de código intermedio puede permitir la aplicación de las medidas de mitigación a prácticamente cualquier tipo de archivo de código intermedio utilizado para generar archivos ejecutables que van desde un simple firmware de archivo binario único hasta sistemas operativos complejos.

Además, las medidas de mitigación de explotación de RAP y ROP, es decir, los segmentos de código agregados comprenden una lógica muy simple y un número bajo y limitado de instrucciones, por lo que requieren recursos de procesamiento y/o almacenamiento muy limitados y usualmente insignificantes para su ejecución y/o almacenamiento. Como tal, las medidas de mitigación de explotación de RAP y ROP aplicadas pueden ser muy adecuadas para dispositivos de gama baja y/o con recursos usualmente limitados.

Además, la aplicación de las medidas de mitigación de explotación de RAP y ROP en los archivos de código intermedio puede no requerir cambios, modificaciones, alteraciones y/o adaptaciones al entorno de desarrollo como pueden requerir los procedimientos existentes. Dado que los entornos de desarrollo que comprenden una o más herramientas, por ejemplo, cadena de herramientas, compilador, enlazador, generador y/o similares pueden ser significativamente complicados, ajustarlos puede requerir recursos y/o tiempos significativos, lo que hace que los procedimientos existentes sean costosos, ineficientes y/o limitados. Por el contrario, las medidas de mitigación de explotación de RAP y ROP aplicadas en el(los) archivo(s) de código intermedio ajustado(s) pueden integrarse fácilmente en el(los) entorno(s) de desarrollo y compilaciones de paquetes de software existentes sin impacto en el entorno del desarrollador. Por ejemplo, las herramientas y/o los procedimientos requeridos para generar y agregar los segmentos de código agregados pueden invocarse agregando una o más entradas (líneas) en uno o más archivos de configuración de compilación y/o enlace, por ejemplo, un archivo de construcción (*makefile)* y/o similar. Además, dado que las medidas de mitigación de la explotación de RAP y ROP se aplican en los archivos de código intermedio y, por lo tanto, no afectan la cadena de herramientas del desarrollador, el desarrollador no necesita recibir capacitación para usar una cadena de herramientas modificada como puede requerirse en los procedimientos existentes.

Antes de explicar en detalle al menos una realización de la invención, debe entenderse que la invención no está necesariamente limitada en su aplicación a los detalles de construcción y la disposición de los componentes y/o procedimientos expuestos en la siguiente descripción y/o ilustrados en los dibujos y/o los Ejemplos. La invención es capaz de otras realizaciones o de ser practicada o llevada a cabo de diversas maneras.

Como apreciará un experto en la técnica, aspectos de la presente invención pueden ser incorporados como un sistema, procedimiento o producto de programa informático. Por consiguiente, aspectos de la presente invención pueden tomar la forma de una realización completamente de hardware, una realización completamente de software (incluyendo firmware, software residente, microcódigo, etc.) o una realización que combina aspectos de software y hardware que pueden denominarse en general en la presente memoria descriptiva como un “circuito”, “módulo” o “sistema”. Además, aspectos de la presente invención pueden tomar la forma de un producto de programa informático incorporado en uno o más medios legibles por ordenador que tienen un código de programa legible por ordenador incorporado en el mismo.

Se puede utilizar cualquier combinación de uno o más medios legibles por ordenador. El medio de almacenamiento legible por ordenador puede ser un dispositivo tangible que puede retener y almacenar instrucciones para su uso por un dispositivo de ejecución de instrucciones. El medio legible por ordenador puede ser un medio de señal legible por ordenador o un medio de almacenamiento legible por ordenador. Un medio de almacenamiento legible por ordenador puede ser, por ejemplo, pero no limitado a, un sistema, aparato o dispositivo electrónico, magnético, óptico, electromagnético, infrarrojo o semiconductor, o cualquier combinación adecuada de los anteriores. Ejemplos más específicos (una lista no exhaustiva) del medio de almacenamiento legible por ordenador incluirían lo siguiente: una conexión eléctrica que tenga uno o más cables, un disquete de ordenador portátil, un disco duro, una memoria de acceso aleatorio (RAM), una memoria de solo lectura (ROM), una memoria de solo lectura programable y borrable (EPROM o memoria Flash), una fibra óptica, una memoria de solo lectura de disco compacto portátil (CD-ROM), un dispositivo de almacenamiento óptico, un dispositivo de almacenamiento magnético o cualquier combinación adecuada de los anteriores. En el contexto de la presente memoria descriptiva, un medio de almacenamiento legible por ordenador puede ser cualquier medio tangible que pueda contener o almacenar un programa para su uso por o en conexión con un sistema, aparato o dispositivo

de ejecución de instrucciones.

Un medio de señal legible por ordenador puede incluir una señal de datos propagada con un código de programa legible por ordenador incorporado en la misma, por ejemplo, en banda base o como parte de una onda portadora. Dicha señal propagada puede adoptar cualquiera de una variedad de formas, incluidas, entre otras, electromagnéticas, ópticas o cualquier combinación adecuada de las mismas. Un medio de señal legible por ordenador puede ser cualquier medio legible por ordenador que no sea un medio de almacenamiento legible por ordenador y que pueda comunicar, propagar o transportar un programa para su uso por o en conexión con un sistema, aparato o dispositivo de ejecución de instrucciones.

El código de Programa Informático que comprende instrucciones de programa legibles por ordenador incorporadas en un medio legible por ordenador puede transmitirse utilizando cualquier medio apropiado, incluidos, entre otros, inalámbricos, cableados, cables de fibra óptica, RF, etc., o cualquier combinación adecuada de los anteriores.

El código de programa para llevar a cabo operaciones para aspectos de la presente invención puede estar escrito en cualquier combinación de uno o más lenguajes de programación, incluyendo un lenguaje de programación orientado a objetos como Java, Smalltalk, C++ o similar y lenguajes de programación procedimentales convencionales, como el lenguaje de programación “C” o lenguajes de programación similares.

El código de programa puede ejecutarse completamente en el ordenador del usuario, parcialmente en el ordenador del usuario, como un paquete de software independiente, parcialmente en el ordenador del usuario y parcialmente en un ordenador remoto o completamente en el ordenador o servidor remoto. En este último escenario, el ordenador remoto puede estar conectado al ordenador del usuario a través de cualquier tipo de red, incluyendo una red de área local (LAN) o una red de área amplia (WAN), o la conexión puede realizarse a un ordenador externo (por ejemplo, a través de Internet utilizando un proveedor de servicios de Internet). El código de programa puede descargarse a los dispositivos de procesamiento/computación respectivos desde un medio de almacenamiento legible por ordenador o a un ordenador externo o dispositivo de almacenamiento externo a través de una red, por ejemplo, Internet, una red de área local, una red de área amplia y/o una red inalámbrica.

Los aspectos de la presente invención se describen en la presente memoria descriptiva con referencia a ilustraciones de diagramas de flujo y/o diagramas de bloques de procedimientos, aparatos (sistemas) y productos de programas informáticos de acuerdo con las realizaciones de la invención. Se entenderá que cada bloque de las ilustraciones de diagramas de flujo y/o diagramas de bloques, y las combinaciones de bloques en las ilustraciones de diagramas de flujo y/o diagramas de bloques, se pueden implementar mediante instrucciones de programa legibles por ordenador.

Los diagramas de flujo y de bloques de las figuras ilustran la arquitectura, la funcionalidad y la operación de posibles implementaciones de sistemas, procedimientos y productos de programas informáticos de acuerdo con diversas realizaciones de la presente invención. A este respecto, cada bloque del diagrama de flujo o de los diagramas de bloques puede representar un módulo, segmento o porción de instrucciones, que comprende una o más instrucciones ejecutables para implementar las funciones lógicas especificadas. En algunas implementaciones alternativas, las funciones indicadas en el bloque pueden ocurrir fuera del orden indicado en las figuras. Por ejemplo, dos bloques mostrados en sucesión pueden, de hecho, ejecutarse sustancialmente de manera concurrente, o los bloques pueden ejecutarse a veces en orden inverso, dependiendo de la funcionalidad involucrada. También se observará que cada bloque de los diagramas de bloques y/o la ilustración del diagrama de flujo, y las combinaciones de bloques en los diagramas de bloques y/o la ilustración del diagrama de flujo, pueden implementarse mediante sistemas basados en hardware de propósito especial que realizan las funciones especificadas o actúan o llevan a cabo combinaciones de instrucciones informáticas y de hardware de propósito especial.

Con referencia ahora a los dibujos, la Figura 1 ilustra un diagrama de flujo de un proceso ejemplar de ajuste de archivos de código intermedio para aplicar protección de dirección de retorno, de acuerdo con algunas realizaciones de la presente invención. Se puede ejecutar un proceso ejemplar 100 para ajustar uno o más archivos de código intermedio, por ejemplo, un archivo de objeto, un archivo binario, un archivo de biblioteca, un archivo de almacenamiento y/o similares generados a partir de uno o más archivos de código fuente por uno o más compiladores para incluir código configurado para aplicar RAP en el o los archivos de código intermedio. El o los archivos de código intermedio que comprenden una pluralidad de rutinas, subrutinas, funciones y/o similares, denominados colectivamente rutina en lo sucesivo, se ajustan antes de ser utilizados para la generación, por ejemplo, construcción, vinculación y/o similares de archivos ejecutables creados para su ejecución por uno o más procesadores de uno o más dispositivos.

Como se ha descrito anteriormente en la presente memoria descriptiva, en tiempo de ejecución un malware puede aplicar una o más vulnerabilidades de seguridad para manipular la pila del procesador o procesadores en un intento de obtener el control sobre el procesador o procesadores utilizando la dirección de retorno tras el

retorno de una o más de las rutinas ejecutadas por el procesador o procesadores para invocar un código malicioso. Para aplicar la RAP, el archivo o archivos de código intermedio pueden ajustarse para incluir una lógica de RAP adaptada para identificar la manipulación de la pila en tiempo de ejecución y evitar que el malware tome el control sobre el flujo de control (ejecución) del procesador o procesadores.

También se hace referencia a la Figura 2, que es una ilustración esquemática de un sistema ejemplar para ajustar los archivos de código intermedio para aplicar protección contra la explotación de la corrupción de memoria, de acuerdo con algunas realizaciones de la presente invención. Un sistema de construcción ejemplar 200, por ejemplo, un ordenador, un servidor, un nodo informático, un grupo de nodos informáticos y/o similares, puede incluir una interfaz de Entrada/Salida (E/S) 202, un procesador o procesadores 204 para ejecutar un proceso tal como el proceso 100 y un almacenamiento 206 para almacenar código y/o datos.

La interfaz de E/S 202 puede incluir una o más interfaces de red para conectarse a una o más redes cableadas y/o inalámbricas, por ejemplo, una red de área local (LAN), una red de área amplia (WAN), una red de área municipal (MAN), una red celular, Internet y/o similares. La interfaz de E/S 202 puede incluir además una o más interfaces, por ejemplo, un bus serie universal (USB), una interfaz de almacenamiento conectable y/o similares para conectarse a uno o más recursos locales, por ejemplo, una unidad de disco externa, otro dispositivo informático y/o similares.

El o los procesadores 204, homogéneos o heterogéneos, pueden incluir uno o más nodos de procesamiento dispuestos para procesamiento en paralelo, como clústeres y/o como uno o más procesadores multinúcleo. El almacenamiento 206 utilizado para almacenar datos y/o código de programa puede incluir uno o más dispositivos de memoria no transitorios, ya sean dispositivos persistentes no volátiles, por ejemplo, un disco duro, una unidad de estado sólido (SSD), un disco magnético, una matriz Flash y/o similares y/o dispositivos volátiles, por ejemplo, un dispositivo de Memoria de Acceso Aleatorio (RAM), una memoria caché y/o similares. El almacenamiento 206 puede comprender además uno o más recursos de almacenamiento de red locales y/o remotos, por ejemplo, un servidor de almacenamiento, un Almacenamiento Conectado a Red (NAS), una unidad de red y/o similares accesibles a través de una o más redes mediante la interfaz de E/S 202.

A través de la interfaz de E/S 202, el sistema de construcción 200 puede obtener, por ejemplo, recibir, buscar y/o recuperar uno o más archivos de código intermedio generados por uno o más compiladores a partir de uno o más archivos de código fuente. El sistema de construcción 200 puede obtener el o los archivos de código intermedio de uno o más recursos de red remotos, por ejemplo, un servidor, un nodo de procesamiento, un servidor de almacenamiento, un NAS, un servicio en la nube, almacenamiento en la nube y/o similares. Adicional y/o alternativamente, a través de la interfaz de E/S 202, el sistema de construcción 200 puede obtener el o los archivos de código intermedio de un recurso de almacenamiento conectado localmente, por ejemplo, un medio de almacenamiento conectable, otro nodo informático y/o similares. El sistema de construcción 200 puede opcionalmente almacenar localmente el o los archivos de código intermedio obtenidos en el almacenamiento 206.

El o los procesadores 204 pueden ejecutar uno o más módulos de software, por ejemplo, un proceso, un *script*, una aplicación, un agente, una utilidad, una herramienta y/o similares, cada uno de los cuales comprende una pluralidad de instrucciones de programa almacenadas en un medio no transitorio, tal como el almacenamiento 206, y ejecutadas por uno o más procesadores, tales como el o los procesadores 204. Por ejemplo, el o los procesadores 204 pueden ejecutar una aplicación de análisis y construcción (constructor) 210 para ajustar el o los archivos de código intermedio para aplicar la RAP. Opcionalmente, el constructor 210 puede estar integrado y/o invocado en uno o más entornos de desarrollo que comprenden una o más herramientas, por ejemplo, una cadena de herramientas, un compilador, un enlazador, un generador y/o similares. Por ejemplo, el constructor 210 puede ser invocado agregando una o más entradas (líneas) en uno o más archivos de configuración de construcción y/o enlace, por ejemplo, un archivo de construcción y/o similares.

Opcionalmente, el sistema de construcción 200 y/o el constructor 210 son proporcionados por uno o más servicios informáticos en la nube, por ejemplo, Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS), Software como Servicio (SaaS) y/o similares proporcionados por una o más infraestructuras y/o servicios en la nube tales como, por ejemplo, Amazon Web Service (AWS), Google Cloud, Microsoft Azure y/o similares.

Como se muestra en 102, el proceso 100 comienza con el constructor 210 obteniendo uno o más archivos de código intermedio, por ejemplo, un archivo de objeto, un archivo binario, un archivo de biblioteca, un archivo de almacenamiento y/o similares pueden ser generados por uno o más compiladores a partir de uno o más archivos de código fuente. El o los archivos de código intermedio pueden obtenerse en uno o más formatos de archivo, por ejemplo, Formato Ejecutable y Enlazable (ELF) y/o similares. El o los archivos de código intermedio pueden utilizarse usualmente para generar, construir y/o enlazar uno o más archivos ejecutables que pueden ser ejecutados por uno o más procesadores. El constructor 210 puede obtener el o los archivos de código intermedio de una o más fuentes, por ejemplo, el o los recursos de red remotos, el dispositivo de almacenamiento conectable y/o el almacenamiento 206.

Como se muestra en 104, el constructor 210 puede aplicar una o más herramientas de análisis de archivos de código intermedio para analizar cada uno de los archivos de código intermedio para identificar todas las rutinas del archivo de código intermedio. Por ejemplo, el constructor 210 puede utilizar "pyelftools", que es una herramienta (biblioteca) de Python para analizar y diseccionar archivos ELF. Utilizando la herramienta "pyelftools", el constructor 210 puede escanear la tabla de símbolos de los archivos intermedios en formato ELF para identificar los símbolos de las rutinas y detectar las direcciones de las rutinas en sus respectivas secciones de código. El constructor 210 puede estar adaptado para analizar los archivos de código intermedio de acuerdo con la arquitectura y el conjunto de instrucciones de los procesadores a los que se dirigen los archivos de código intermedio, por ejemplo, ARM, ARM-Thumb, x86, x86-64, power ISA y/o similares.

Por ejemplo, el constructor 210 puede identificar las rutinas iterando sobre una tabla de símbolos del archivo de código intermedio para identificar y mapear los símbolos generados por los compiladores para cada una de las rutinas en el archivo de código intermedio. Por ejemplo, en la arquitectura ARM-Thumb, cada archivo de objeto ".o" (intermedio) se implementa en el formato de archivo ELF. El constructor 210 puede identificar los nombres de los símbolos de las rutinas iterando sobre las entradas en la sección de "tabla de símbolos" asignada con la extensión ".symtab" en el estándar ELF. La tabla de símbolos comprende información requerida para localizar y reubicar las definiciones y referencias simbólicas del programa. Cada entrada de símbolo asociada con una de las rutinas en la tabla de símbolos se caracteriza por tener un campo de tamaño distinto de cero, un campo de tipo "FUNC" y un campo de índice que hace referencia a una de las secciones de "código" en el archivo de objeto ".o" (intermedio). Una o más de las entradas de la tabla de símbolos pueden incluir además un campo de enlace "GLOBAL" y/o un campo de enlace "LOCAL".

El constructor 210 puede analizar además cada uno de los archivos de código intermedio para identificar una dirección de inicio de cada una de las rutinas en el archivo de código intermedio. Por ejemplo, en la arquitectura ARM-Thumb, el valor del símbolo "Función" (rutina) en la tabla de símbolos es "inicio de la función (rutina) + 1" para especificar que la función contiene código Thumb. En tal caso, y opcionalmente en casos similares para otras arquitecturas de procesador, el constructor 210 puede extraer el "valor" de la rutina de la tabla de símbolos de un archivo de objeto ".o" (intermedio) y realizar la operación matemática opuesta para extraer la dirección de inicio real de la rutina, es decir, "valor - 1".

Como se muestra en 106, el constructor 210 analiza una o más de la pluralidad de rutinas en cada uno de los archivos de código intermedio para identificar la instrucción de inserción en pila de dirección de retorno y una o más correspondientes instrucciones de extracción de pila de dirección de retorno. El constructor 210 puede analizar las rutinas extrayendo primero su código binario (de máquina) y desensamblando el código de máquina para proporcionar el código ensamblador de las rutinas. El constructor 210 analiza o analiza más a fondo las rutinas para identificar una o más instrucciones de inserción en pila y las respectivas instrucciones de extracción de pila a punteros de elementos de datos, por ejemplo, una variable, una estructura, una matriz y/o similares.

El constructor 210 puede extraer el código binario de la(s) rutina(s) del archivo de código intermedio utilizando la información en la entrada de la tabla de símbolos correspondiente a cada rutina. El constructor 210 puede aplicar luego uno o más procedimientos, técnicas y/o herramientas de desensamblado de archivos de código intermedio para desensamblar el código binario de la rutina en instrucciones de ensamblaje y analizar las instrucciones de ensamblaje. Por ejemplo, el constructor 210 puede utilizar la herramienta "pyelftools" para extraer el código de cada rutina como código de máquina. El constructor 210 puede aplicar una o más herramientas, por ejemplo, "Capstone Disassembler" y/o similares, para desensamblar el código de máquina extraído.

La sintaxis y/o los códigos de operación de las instrucciones de inserción en pila de dirección de retorno y de extracción de pila de dirección de retorno pueden variar entre las arquitecturas de procesador y/o los conjuntos de instrucciones. Por lo tanto, el constructor 210 puede estar adaptado para analizar las rutinas con el fin de identificar las instrucciones de inserción en pila de dirección de retorno y extracción de pila de dirección de retorno de acuerdo con la sintaxis de la arquitectura de procesador y el conjunto de instrucciones seleccionado para generar el o los archivos de código intermedio. Por ejemplo, suponiendo que la arquitectura de procesador es la arquitectura ARM-Thumb, el constructor 210 puede buscar cada rutina desensamblada para identificar las instrucciones de inserción en pila implementadas por uno o más códigos de operación ARM-Thumb, como, por ejemplo, PUSH {RegisterName1,...,RegisterNameN}, STMFD SP!, {RegisterName1,...,RegisterNameN}, STMIA SP!, {RegisterName1,...,RegisterNameN} y más. En algunas de las arquitecturas de procesador, por ejemplo, la arquitectura x86, la dirección de retorno puede insertarse automáticamente en la pila sin que exista ninguna instrucción de inserción en pila específica en el código. En tal caso, el constructor 210 puede identificar la instrucción que inserta el puntero base hacia la pila.

Además, el constructor 210 puede estar adaptado para iterar a través de un número predefinido de instrucciones al inicio de una o más de las rutinas para detectar las instrucciones de inserción en pila de dirección de retorno. En caso de que el constructor 210 no encuentre una instrucción de inserción en pila de dirección de retorno al inicio de una determinada rutina, el constructor 210 puede determinar que la determinada rutina es una rutina "hoja" que no llama (ramifica) a otras rutinas y, por lo tanto, no incluye operaciones de inserción en pila y

extracción de pila correspondientes. Como tal, la determinada rutina puede no ser susceptible a la explotación de desbordamiento de la pila y el constructor 210 puede marcar la determinada rutina en consecuencia, es decir, no es necesario aplicar las medidas RAP (lógica) en la determinada rutina.

Si bien tienen un único punto de entrada, una o más de las rutinas pueden incluir múltiples puntos finales en los que la rutina puede regresar a la rutina que llama al ramificarse a la dirección de retorno insertada en la pila por medio de la instrucción de inserción en pila de dirección de retorno en el punto de entrada de la rutina. Por lo tanto, el constructor 210 puede analizar el código ensamblador de la(s) rutina(s) para identificar la instrucción de extracción de pila de dirección de retorno en todos los puntos finales de la rutina.

El constructor 210 puede analizar primero la instrucción de inserción en pila de dirección de retorno identificada para identificar el lugar en la pila que ocupará la dirección de retorno. El constructor 210 puede entonces iterar a través de las instrucciones de ensamblaje de rutinas para identificar las instrucciones configuradas para utilizar la dirección de retorno desde la pila. El constructor 210 puede lograr esto manteniendo un valor del puntero de pila actual y su distancia desde la posición de la dirección de retorno y actualizando el valor del puntero de pila actual con cada instrucción que afecte al puntero de pila. El constructor 210 puede, de este modo, detectar cada instrucción que utilice la dirección de retorno insertada y puede identificar esta instrucción como una instrucción de extracción de pila de dirección de retorno.

Una o más de las rutinas pueden incluir ramificaciones internas en las que se puede indicar al procesador que se ramifique a otra ubicación dentro de la misma rutina. Por lo tanto, el constructor 210 puede estar adaptado para analizar más a fondo las ubicaciones objetivo de ramificación de las ramificaciones internas para verificar posibles instrucciones de extracción de pila de dirección de retorno.

Naturalmente, el constructor 210 mantiene un registro de las rutinas analizadas y/o parte de las mismas para evitar volver a analizar una sección de código ya analizada.

El constructor 210 puede marcar las ubicaciones identificadas de las instrucciones de inserción en pila de dirección de retorno y las instrucciones de extracción de pila de dirección de retorno identificadas en la(s) rutina(s) relevante(s) del(de los) archivo(s) de código intermedio, es decir, la(s) rutina(s) en la(s) que están presentes dichas instrucciones.

Como se muestra en 108, el constructor 210 puede ajustar el(los) archivo(s) de código intermedio ajustando una o más de las rutinas en las que se identificaron y marcaron las instrucciones de inserción/extracción en/de pila de dirección de retorno para incluir las medidas de RAP (lógica). Específicamente, el constructor 210 ajusta cada una de dichas rutinas reemplazando la instrucción de inserción en pila de dirección de retorno identificada con un segmento de código de prólogo y reemplazando cada instrucción de extracción de pila de dirección de retorno identificada con un segmento de código de epílogo correspondiente.

El constructor 210 puede agregar el segmento de código de prólogo y/o el(los) segmento(s) de código de epílogo para reemplazar la instrucción de inserción en pila de dirección de retorno y/o la(s) instrucción(es) de extracción de pila de dirección de retorno respectivamente utilizando uno o más procedimientos, técnicas y/o implementaciones de codificación. Por ejemplo, el constructor 210 puede reemplazar cada una de las instrucciones de inserción en pila de dirección de retorno y/o las instrucciones de extracción de pila de dirección de retorno con una función de ramificación de trampolín configurada para invocar el segmento de código agregado respectivo durante las operaciones de inserción y extracción de dirección. Antes de insertar los segmentos de código agregados en el(los) archivo(s) de código intermedio ajustado, el constructor 210 puede verificar que haya suficientes recursos disponibles, por ejemplo, espacio de almacenamiento y/o similares para alojar el(los) segmento(s) de código agregados. Sin embargo, dado que los segmentos de código agregados comprenden usualmente una lógica muy simple, la huella de los segmentos de código agregados puede ser significativamente pequeña, por lo que no presenta limitaciones para integrarlos en el(los) archivo(s) de código intermedio ajustado.

El segmento de código de prólogo que reemplaza la instrucción de inserción en pila de dirección de retorno está configurado para alterar la pila después de que la dirección de retorno se inserte en la pila, creando así una firma única que puede ser validada posteriormente por el(los) segmento(s) de código de prólogo respectivo(s) que reemplaza(n) la(s) instrucción(es) de extracción de pila de dirección de retorno. Durante el tiempo de ejecución, cuando uno o más procesadores ejecutan el(los) archivo(s) ejecutable(s) generado(s) utilizando el(los) archivo(s) de código intermedio ajustado(s), el segmento de código de epílogo puede acceder a la pila para recuperar la dirección de retorno insertada en la pila por el respectivo segmento de código de prólogo e intentar validar la alteración de la pila realizada por el respectivo segmento de código de prólogo. En caso de que el segmento de código de epílogo pueda validar la alteración de la pila, el segmento de código de epílogo puede permitir que el o los procesadores se ramifiquen a la dirección de retorno recuperada. Sin embargo, en caso de que el segmento de código de epílogo no pueda validar la alteración de la pila, el segmento de código de epílogo puede hacer que el o los procesadores inicien una o más acciones predefinidas. Las acciones predefinidas pueden incluir, por

ejemplo, bloquear la ejecución del o los procesadores, detener la ejecución del o los procesadores, hacer que el o los procesadores se ramifiquen a una dirección predefinida, evitar que el o los procesadores ejecuten al menos una instrucción de código potencialmente malicioso dirigida por una pila corrupta, generar una o más indicaciones y/o alertas de alteración de pila inválida y/o similares. Las acciones predefinidas pueden seleccionarse de acuerdo con uno o más parámetros del flujo de ejecución, por ejemplo, la arquitectura del procesador, una severidad y/o criticidad de cada rutina, un parámetro definido por el usuario y/o similares.

El constructor 210 puede aplicar uno o más procedimientos, técnicas y/o implementaciones para configurar el segmento de código de prólogo y el segmento de código de epílogo para aplicar la alteración de la pila y recuperar la alteración de la pila respectivamente utilizando uno o más procedimientos, técnicas y/o implementaciones.

En una primera implementación ejemplar, el segmento de código de prólogo puede estar configurado para alterar la pila modificando la dirección de retorno insertada en la pila de una manera reversible de modo que el o los respectivos segmentos de código de epílogo puedan recuperar la dirección de retorno insertada originalmente a partir de la dirección de retorno alterada. El segmento de código de prólogo puede estar configurado para leer la dirección de retorno insertada en la pila, alterar la dirección de retorno de una manera reversible, por ejemplo, cifrar la dirección de retorno e introducir la dirección de retorno alterada de nuevo en la pila. El segmento de código de prólogo puede alterar la dirección de retorno aplicando uno o más operadores matemáticos, operadores lógicos y/o similares a la dirección de retorno insertada originalmente. Por ejemplo, el segmento de código de prólogo puede estar configurado para alterar la dirección de retorno aplicando una operación XOR a la dirección de retorno con un valor constante seleccionado aleatoriamente. El valor constante seleccionado aleatoriamente puede generarse, por ejemplo, utilizando un generador de números aleatorios. En otro ejemplo, el valor constante seleccionado aleatoriamente puede seleccionarse aleatoriamente de un conjunto de datos que comprende una cantidad extremadamente grande de valores aleatorios.

El o los respectivos segmentos de código de epílogo pueden estar configurados para recuperar y/o descifrar la dirección de retorno insertada originalmente a partir de la dirección de retorno alterada invirtiendo la operación realizada por el respectivo segmento de código de prólogo. Por ejemplo, suponiendo que el segmento de código de prólogo está configurado para realizar una operación XOR de la dirección de retorno con el valor constante seleccionado aleatoriamente, el o los segmentos de código de epílogo pueden estar configurados para realizar una operación XOR de la dirección de retorno alterada con el mismo valor constante (utilizado por el segmento de código de prólogo) para recuperar la dirección de retorno insertada originalmente. El o los respectivos segmentos de código de epílogo pueden estar configurados además para validar la dirección de retorno recuperada antes de ramificarse a la dirección de retorno recuperada (extraída) de la pila. Por ejemplo, uno o más de los respectivos segmentos de código de epílogo pueden verificar que la dirección de retorno recuperada se encuentra dentro de un intervalo de direcciones válidas del código de programa codificado en el o los archivos de código intermedio.

Durante el tiempo de ejecución, el valor constante seleccionado aleatoriamente que sirve como firma única para la pila modificada puede ser validado por el o los segmentos de código de epílogo antes de la ramificación a la dirección de retorno recuperada de la pila. Para hacer que los segmentos de código de prólogo y epílogo agregados sean independientes de la arquitectura, el conjunto de instrucciones y/o el sistema operativo del procesador o procesadores que ejecutan el(los) archivo(s) ejecutable(s) creado(s) utilizando el(los) archivo(s) de código intermedio ajustado(s), el valor constante seleccionado aleatoriamente utilizado para la operación XOR puede definirse como una constante global para garantizar la seguridad de los subprocesos. Sin embargo, para aumentar la seguridad, el valor constante seleccionado aleatoriamente puede ser transitorio y reemplazarse con frecuencia, por ejemplo, seleccionado aleatoriamente durante cada secuencia de inicio (arranque) del procesador que ejecuta el archivo o archivos ejecutables, durante la primera invocación del segmento de código de prólogo y/o similares.

En una segunda implementación ejemplar, el segmento de código de prólogo puede estar configurado para alterar la pila utilizando canarios de pila, es decir, introduciendo un marcador (canario) en la pila en una ubicación adyacente a la ubicación de dirección de retorno insertada de modo que el o los respectivos segmentos de código de epílogo puedan validar el valor del marcador insertado en la pila. La ubicación adyacente en la pila en la que se introduce el marcador puede preceder a la ubicación de dirección de retorno insertada en la pila o suceder a la ubicación de dirección de retorno insertada. Por ejemplo, en caso de que la pila emplee una implementación de pila descendente completa en la que la dirección de pila decrece con cada instrucción de inserción, el marcador puede ser insertado en una ubicación de dirección que precede a la dirección de la dirección de retorno insertada. En otro ejemplo, en caso de que la pila emplee una implementación de pila ascendente completa en la que la dirección de pila se incrementa con cada instrucción de inserción, el marcador puede introducirse en una ubicación de dirección posterior a la dirección de la dirección de retorno insertada. En concreto, el segmento de código de prólogo puede estar configurado para insertar un marcador de valor constante en la ubicación adyacente en la pila, mientras que el o los respectivos segmentos de código de epílogo pueden estar configurados para verificar el valor del marcador insertado en la pila por el segmento de código de

prólogo. El marcador de valor constante puede generarse, por ejemplo, utilizando un generador de números aleatorios, como, por ejemplo, un Generador de Números Aleatorios Verdaderos (TRNG) de hardware, un Generador Pseudoaleatorio de software (PRNG) y/o similares. En otro ejemplo, el marcador de valor constante puede seleccionarse aleatoriamente de un conjunto de datos que comprende una cantidad extremadamente grande de valores aleatorios.

Como se describe para el valor constante seleccionado aleatoriamente utilizado para la operación XOR en la implementación de alteración de pila ejemplar anterior, a fin de hacer que los segmentos de código de prólogo y epílogo agregados sean independientes de la arquitectura, el conjunto de instrucciones y/o el sistema operativo del procesador o procesadores, el marcador de valor constante insertado por el segmento de código de prólogo y verificado por el segmento o segmentos de código de epílogo puede definirse como una constante global para garantizar la seguridad del subproceso. De manera similar, para aumentar la seguridad, el marcador de valor constante puede ser transitorio y reemplazarse con frecuencia, por ejemplo, seleccionado aleatoriamente durante cada secuencia de inicio (arranque) del procesador que ejecuta el archivo o archivos ejecutables, durante la primera invocación del segmento de código de prólogo y/o similar.

Durante el tiempo de ejecución, el marcador de valor constante introducido que sirve como firma única para la pila alterada puede ser validado por el segmento o segmentos de código de epílogo antes de ramificarse a la dirección de retorno recuperada de la pila. Dado que el marcador se inserta en la pila, el diseño de la pila establecido durante la fase de compilación puede cambiar, ya que la distancia de algunos parámetros de la rutina ajustada desde el puntero de pila puede cambiar. Por lo tanto, el constructor 210 puede analizar el código desensamblado de la rutina ajustada para buscar instrucciones que accedan a estos parámetros en la pila y modificar estas instrucciones para acceder al desplazamiento correcto en la pila. Por ejemplo, suponiendo que el archivo o archivos de código intermedio se compilan para la arquitectura ARM y el conjunto de instrucciones, el constructor 210 puede buscar instrucciones 'ldr Rn, [sp #desplazamiento]' donde 'sp' es el puntero de pila. El constructor 210 puede entonces modificar el valor '#desplazamiento' de acuerdo con el nuevo diseño de la pila, es decir, de acuerdo con la nueva distancia de cada uno de los parámetros desde el puntero de pila.

Una o más de las instrucciones de inserción en pila de dirección de retorno identificadas y marcadas en una o más de las rutinas en el archivo o archivos de código intermedio pueden estar asociadas con la inserción de uno o más elementos de datos en la pila. Por lo tanto, el constructor 210 puede configurar el o los segmentos de código de prólogo agregados para reemplazar dichas instrucciones de inserción en pila de dirección de retorno en consecuencia para insertar el o los elementos de datos en la pila. Por ejemplo, suponiendo que se aplica la función de ramificación de trampolín para insertar un determinado segmento de código de prólogo en una determinada rutina que comprende una o más instrucciones de inserción en pila de datos. El constructor 210 puede configurar el determinado segmento de código de prólogo para separar la instrucción de inserción en pila de dirección de retorno de la o las instrucciones de inserción en pila de datos de modo que la o las instrucciones de inserción en pila de datos se lleven a cabo después de completarse la ejecución del determinado segmento de código de prólogo. De manera complementaria, la o las instrucciones de extracción de pila de dirección de retorno correspondientes a dichas instrucciones de inserción en pila de dirección de retorno pueden recuperar los elementos de datos introducidos mientras recuperan la dirección de retorno de la pila. Por lo tanto, el constructor 210 puede configurar el o los respectivos segmentos de código de epílogo agregados para reemplazar dichas instrucciones de extracción de pila de dirección de retorno en consecuencia para recuperar el o los elementos de datos introducidos en la pila por el o los segmentos de código de prólogo correspondientes. Continuando con el ejemplo presentado, suponiendo que se aplica la función de ramificación de trampolín para insertar un determinado segmento de código de epílogo correspondiente al determinado segmento de código de prólogo insertado en la determinada rutina que comprende una o más instrucciones de inserción en pila de datos. El constructor 210 puede configurar el determinado segmento de código de epílogo para separar la(s) instrucción(es) de extracción de pila de dirección de retorno de la(s) instrucción(es) de extracción de pila de datos de modo que la(s) instrucción(es) de extracción de pila de datos se realicen antes de la ejecución del determinado segmento de código de epílogo.

Como se muestra en 110, el constructor 210 modifica datos, instrucción(es), tabla(s) de símbolos y/o uno o más atributos del(de los) archivo(s) de código intermedio afectados por el ajuste del(de los) archivo(s) de código intermedio realizado para incluir el(los) segmento(s) de código de prólogo, el(los) segmento(s) de código de epílogo y/o los marcadores.

Por ejemplo, el diseño de una o más de las rutinas ajustadas en el archivo o archivos de código intermedio, tal como se define durante la compilación del archivo o archivos de código intermedio, puede cambiar debido a la inserción de los segmentos de código y/o marcadores agregados, cambiando así las ubicaciones relativas de una o más instrucciones y/o elementos de datos en la rutina o rutinas ajustadas. Por lo tanto, el constructor 210 puede analizar el código desensamblado de la rutina o rutinas para buscar instrucciones y/o elementos de datos que comprendan referencias y/o punteros a otras instrucciones y/u otros elementos de datos. El constructor 210 puede ajustar las referencias y/o punteros detectados y actualizarlos de acuerdo con el nuevo diseño de la rutina o rutinas después de la inserción de los segmentos de código y/o marcadores agregados. Por ejemplo,

suponiendo que el archivo o archivos de código intermedio se compilan para la arquitectura ARM y el conjunto de instrucciones, el constructor 210 puede buscar 'LDR Rn, [pc #desplazamiento]' que puede verse afectado por el cambio del diseño de la rutina. El ajustador puede entonces modificar (actualizar) el '#desplazamiento' de acuerdo con el nuevo diseño de la rutina ajustada para señalar la ubicación correcta con respecto al 'pc' que es el contador de programa.

En otro ejemplo, el constructor 210 puede modificar la tabla de símbolos de la(s) rutina(s) ajustada(s) para reflejar los cambios aplicados a la(s) rutina(s) ajustada(s) por la inserción de los segmentos de código y/o marcadores agregados. Por ejemplo, el constructor 210 puede actualizar la tabla de símbolos para incluir los símbolos de los segmentos de código agregados. El constructor 210 puede modificar además la tabla de símbolos para reflejar las ubicaciones de dirección de la(s) rutina(s) ajustada(s) que pueden haber cambiado debido a la inserción de los segmentos de código y/o marcadores agregados.

Además, la inserción de los segmentos de código agregados, así como la inserción de los marcadores pueden inflar el tamaño del archivo de código intermedio respectivo. Por lo tanto, el constructor 210 puede ajustar el(los) archivo(s) intermedio(s) ajustado(s) para modificar uno o más atributos del(de los) archivo(s) de código intermedio ajustado(s), por ejemplo, el tamaño del archivo y/o similares. Por ejemplo, suponiendo que el o los archivos de código intermedio están en formato ELF, el constructor 210 puede ajustar el encabezado del o los archivos ELF para reflejar los nuevos desplazamientos y alineaciones en la o las rutinas del o los archivos de código intermedio después de la inserción de los segmentos de código y/o marcadores agregados.

El constructor 210 puede incluir el código agregado y/o el o los marcadores en el o los archivos intermedios ajustados. Adicional y/o alternativamente, el constructor 210 puede incluir el código agregado y/o el o los marcadores en uno o más archivos de código intermedio adicionales. El o los archivos de código intermedio adicionales pueden proporcionarse junto con el o los archivos de código intermedio ajustados para la generación, construcción y/o vinculación del o los archivos ejecutables.

El constructor 210 también puede verificar que las ramificaciones a los segmentos de código agregados implementados, por ejemplo, utilizando las funciones de ramificación de trampolín sean válidas para la construcción y/o vinculación después de que el(los) archivo(s) de código intermedio se ajusten para reflejar los cambios implicados por la inserción de los segmentos de código agregados y/o los marcadores. Por ejemplo, el constructor 210 puede agregar una entrada de reubicación para cada ramificación a uno de los segmentos de código agregados en los que la entrada de reubicación comprende un nombre predefinido. La entrada de reubicación puede agregarse a la sección de reubicación que describe las reubicaciones para la(s) sección(es) de código que contienen la(s) función(es) de ramificación. Si no existe dicha sección de código, la entrada de reubicación puede crearse y agregarse a las secciones existentes en el archivo intermedio. Los segmentos de código agregados pueden compilarse con los mismos nombres predefinidos en uno o más de los archivos de código intermedio adicionales creados para incluir los segmentos de código agregados.

Como se muestra en 112, el constructor 210 puede generar el o los archivos de código intermedio ajustados que se pueden usar para generar, construir y/o vincular uno o más archivos ejecutables que se pueden ejecutar por uno o más procesadores. Por ejemplo, el constructor 210 puede transmitir el o los archivos de código intermedio a uno o más de los recursos en red remotos que pueden usar una o más aplicaciones, herramientas y/o similares, por ejemplo, un enlazador, un generador de código y/o similares para crear el o los archivos ejecutables a partir del o los archivos de código intermedio ajustados para crear el o los archivos ejecutables. En otro ejemplo, el constructor 210 puede almacenar el o los archivos de código intermedio en el almacenamiento 206 desde el que el o los archivos de código intermedio se pueden recuperar por una o más aplicaciones, herramientas y/o similares, por ejemplo, un enlazador, un generador de código y/o similares para crear el o los archivos ejecutables. En otro ejemplo, el constructor 210 puede almacenar el(los) archivo(s) de código intermedio en uno o más de los dispositivos de almacenamiento conectables que pueden estar conectados a otro sistema donde el(los) archivo(s) de código intermedio pueden ser recuperados por una o más aplicaciones, herramientas y/o similares, por ejemplo, un enlazador, un generador de código y/o similares para crear el(los) archivo(s) ejecutables.

De acuerdo con algunas realizaciones de la presente invención, se proporcionan procedimientos, sistemas y productos de programas informáticos para ajustar uno o más de los archivos de código intermedio para evitar la explotación de ROP. Como se ha descrito anteriormente en la presente memoria descriptiva, en tiempo de ejecución, una parte maliciosa puede explotar ROP para desviar el procesador o los procesadores para invocar uno o más dispositivos integrados en una o más rutinas en el archivo o los archivos intermedios en un intento de obtener el control sobre el procesador o los procesadores. A fin de aplicar la prevención de la explotación de ROP, el archivo o los archivos de código intermedio pueden ajustarse para incluir una lógica preventiva de explotación de ROP adaptada para identificar operaciones de ramificación no válidas que pueden invocar potencialmente el dispositivo o los dispositivos. Dado que en tiempo de ejecución la parte maliciosa puede manipular solo recursos de memoria volátil, la lógica de prevención de explotación de ROP puede aplicarse usualmente a operaciones de ramificación indirecta que pueden recuperar su dirección de ramificación de

recursos de memoria volátil (por ejemplo, pila, montón, registro, etc.). Las operaciones de ramificación directa en las que la dirección de ramificación está codificada pueden ser inmunes a la explotación de ROP, ya que la dirección de ramificación no se recupera de un recurso de memoria volátil. Por lo tanto, la lógica de prevención de explotación de ROP puede no ser necesaria y, por lo tanto, no aplicarse para operaciones de ramificación directa.

Ahora se hace referencia a la Figura 3, que es un diagrama de flujo de un proceso ejemplar de ajuste de archivos de código intermedio para aplicar protección contra la explotación de programación orientada al retorno, de acuerdo con algunas realizaciones de la presente invención. Un proceso ejemplar 300 puede ser ejecutado por un ajustador tal como el constructor 210 ejecutado por un sistema de construcción tal como el sistema de construcción 200 para ajustar uno o más de los archivos de código intermedio para incluir código configurado para evitar la ROP.

Como se muestra en 302, el proceso 300 comienza con el constructor 210 obteniendo uno o más archivos de código intermedio como se describe en la etapa 102 del proceso 100.

Como se muestra en 304, el constructor 210 analiza cada uno de los archivos de código intermedio para identificar todas las rutinas del archivo de código intermedio como se describe en la etapa 104 del proceso 100.

Como se muestra en 306, el constructor 210 analiza una o más de la pluralidad de rutinas en cada uno de los archivos de código intermedio para identificar ramificaciones indirectas en las rutinas como se describe en la etapa 106 del proceso 100.

Como se muestra en 308, el constructor 210 puede ajustar los archivos de código intermedio ajustando una o más de las rutinas en las que se identificaron y marcaron ramificaciones indirectas para incluir las medidas de prevención de explotación de ROP (lógica). En concreto, el constructor 210 ajusta cada una de dichas rutinas sustituyendo la ramificación indirecta identificada por una invocación de un respectivo segmento de código de verificación. El constructor 210 puede aplicar uno o más procedimientos, técnicas y/o implementaciones de codificación para sustituir la ramificación indirecta por la invocación del segmento de código de verificación. Por ejemplo, el constructor 210 puede sustituir una o más instrucciones de ramificación indirecta por ramificaciones directas respectivas que apunten hacia la ubicación de dirección del segmento o segmentos de código de verificación agregados de modo que durante el tiempo de ejecución, en lugar de ejecutar la ramificación indirecta, se invoque y ejecute el segmento de código de verificación. El segmento o segmentos de código de verificación se pueden implementar, por ejemplo, utilizando una función de trampolín. Como se describe en la etapa 108, antes de insertar el segmento o segmentos de código de verificación agregados en el archivo o archivos de código intermedio ajustados, el constructor 210 puede verificar que haya suficientes recursos disponibles, por ejemplo, espacio de almacenamiento y/o similares para alojar los segmentos de código agregados. Sin embargo, dado que los segmentos de código agregados comprenden usualmente una lógica muy simple, la huella de los segmentos de código agregados puede ser significativamente pequeña, por lo que no presenta limitaciones para integrarlos en el archivo o archivos de código intermedio ajustados.

El constructor 210 puede configurar cada segmento de código de verificación para obtener el símbolo de rutina y/o la dirección a la que apunta la respectiva instrucción de ramificación indirecta. El constructor 210 puede configurar además el segmento de código de verificación para verificar que el símbolo de rutina y/o la dirección a la que apunta la ramificación indirecta respectiva que apunta hacia una ubicación de dirección que es el inicio de una rutina válida, es decir, una de la pluralidad de rutinas del archivo o archivos de código intermedio.

Se pueden aplicar una o más implementaciones y/o técnicas para transferir al segmento de código de verificación la dirección de ramificación, es decir, el símbolo y/o la dirección de la rutina a la que apunta una o más de las instrucciones de ramificación indirecta. Por ejemplo, la dirección de ramificación se puede introducir en la pila de modo que, cuando se invoque, el segmento de código de verificación pueda recuperar (leer, extraer, etc.) la dirección de ramificación de la pila. Para ello, el constructor 210 puede agregar una instrucción de inserción en pila para introducir la dirección de ramificación incluida en una respectiva instrucción de ramificación indirecta y colocar la instrucción de inserción en pila agregada antes de la instrucción de ramificación directa insertada para sustituir la respectiva instrucción de ramificación indirecta. En otro ejemplo, la dirección de ramificación se puede escribir en un registro predeterminado del procesador o procesadores que ejecutan el código de modo que, cuando se invoque, el segmento de código de verificación pueda acceder al registro predeterminado para recuperar la dirección de ramificación. Para este fin, el constructor 210 puede agregar una instrucción de escritura para escribir la dirección de ramificación incluida en una respectiva instrucción de ramificación indirecta en el registro predeterminado y colocar la instrucción de escritura agregada antes de la instrucción de ramificación directa insertada para reemplazar la respectiva instrucción de ramificación indirecta. En otro ejemplo, la dirección de ramificación puede escribirse en una variable predefinida (y/o estructura) almacenada en la memoria del procesador que ejecuta el código de modo que, cuando se invoque, el segmento de código de verificación pueda acceder a la variable predefinida para recuperar la dirección de ramificación. Para este fin, el constructor 210 puede inicializar la variable predefinida y agregar una instrucción de escritura

para escribir la dirección de ramificación incluida en una respectiva instrucción de ramificación indirecta en la variable predefinida y colocar la instrucción de escritura agregada antes de la instrucción de ramificación directa insertada para reemplazar la respectiva instrucción de ramificación indirecta.

Durante el tiempo de ejecución, cuando uno o más procesadores ejecutan el o los archivos ejecutables generados utilizando el o los archivos de código intermedio ajustados, el segmento de código de verificación puede obtener primero el símbolo de rutina y/o la dirección (dirección de ramificación) a la que apunta la respectiva ramificación indirecta y verificar que apunta al inicio (dirección) de una rutina válida. En caso de que el segmento de código de verificación sea capaz de validar que la ramificación se lleva al principio de una rutina válida, el segmento de código de verificación puede permitir que el procesador o los procesadores se ramifiquen a la dirección de ramificación. Sin embargo, en caso de que el segmento de código de verificación no pueda validar que la ramificación se lleva al principio de una rutina válida, el segmento de código de verificación puede determinar que la ramificación indirecta es inválida y puede hacer que el procesador o los procesadores inicien una o más de las acciones predefinidas. Las acciones predefinidas pueden incluir, por ejemplo, bloquear la ejecución del procesador o los procesadores, detener la ejecución del procesador o los procesadores, hacer que el procesador o los procesadores se ramifiquen a una dirección predefinida, hacer que el procesador o los procesadores impidan la ejecución del dispositivo o los dispositivos, hacer que el procesador o los procesadores generen una o más indicaciones de la ramificación o las ramificaciones inválidas y/o similares. Las acciones predefinidas pueden seleccionarse de acuerdo con uno o más parámetros del flujo de ejecución, por ejemplo, la arquitectura del procesador, una severidad y/o criticidad de cada rutina, un parámetro definido por el usuario y/o similares.

El constructor 210 puede aplicar uno o más procedimientos, técnicas y/o implementaciones para configurar el segmento de código de verificación para verificar que el símbolo y/o la dirección de rutina apuntados por la respectiva ramificación indirecta sea el inicio de una rutina válida.

En una primera implementación ejemplar, el constructor 210 puede asociar cada una de la pluralidad de rutinas de cada uno de los archivos de código intermedio con un código único insertado (agregado) en una ubicación de dirección que precede a la ubicación de dirección del inicio de la rutina respectiva. Específicamente, el constructor 210 asocia el código único con cada rutina a la que se identifica una ramificación indirecta durante el análisis del código ensamblador de la etapa 306. Opcionalmente, el constructor 210 puede asociar la pluralidad de rutinas con un código único común, es decir, similar. Por ejemplo, el constructor 210 puede insertar un código único en la dirección inmediatamente anterior a la dirección de inicio de una o más rutinas válidas. El constructor 210 puede seleccionar la dirección exacta para la inserción del código único de acuerdo con la arquitectura y/o el conjunto de instrucciones del procesador seleccionado. Por ejemplo, suponiendo que la arquitectura del procesador es de 32 bits e implementa un conjunto de instrucciones de 32 bits con una granularidad de “palabra” nativa de 4 bytes, el constructor 210 puede insertar el código único en la dirección “dirección de inicio - 4”. Además, el código único utilizado por el constructor 210 puede no ser parte del conjunto de instrucciones del procesador o procesadores seleccionados para evitar que el procesador o procesadores lo interpreten como una instrucción válida y, por lo tanto, impidan alterar el flujo de ejecución del procesador o procesadores. Además, el código único se selecciona para que no sea parte del conjunto de instrucciones del procesador o procesadores seleccionados con el fin de evitar la aparición irrelevante del código único que no es insertado por el constructor 210 sino por el compilador que generó el archivo o archivos de código intermedio.

El constructor 210 puede configurar el segmento de código de verificación para recuperar primero el código único de la dirección designada, por ejemplo, la dirección que precede a la dirección de ramificación indicada por la instrucción de ramificación indirecta. Para continuar con el ejemplo anterior, en caso de que el código único se inserte en la dirección ‘dirección de inicio - 4’, el constructor 210 puede configurar el segmento de código de verificación para acceder primero a la ‘dirección de inicio - 4’ para recuperar el código único. El constructor 210 puede configurar además el segmento de código de verificación para validar el valor del código único y determinar en consecuencia si la dirección de la ramificación apunta a la dirección de inicio de una de las rutinas válidas.

Durante el tiempo de ejecución, en caso de que el segmento de código de verificación valide positivamente el valor del código único, el segmento de código de verificación puede determinar que la ramificación indirecta está apuntando al inicio de una rutina válida. Sin embargo, en caso de que el segmento de código de verificación no pueda validar correctamente el valor del código único, el segmento de código de verificación puede determinar que la ramificación indirecta no es válida ya que no está apuntando al inicio de una rutina válida.

En una segunda implementación ejemplar, el constructor 210 puede construir un conjunto de datos, por ejemplo, una tabla, una lista y/o similar para mapear la dirección de inicio de cada una de la pluralidad de rutinas, específicamente, la dirección de inicio de cada rutina a la que se identifica una ramificación indirecta durante el análisis del código ensamblador de la etapa 306. Como tal, el conjunto de datos puede incluir una o más entradas, cada una asociada con una de las rutinas mapeadas y que almacena la dirección de inicio de la rutina respectiva. Las direcciones absolutas de las rutinas, en particular las direcciones de inicio de las rutinas, pueden

no estar disponibles para los archivos de código intermedio, ya que usualmente pueden establecerse durante la etapa de generación de los archivos ejecutables, es decir, después de la construcción y/o vinculación. Para superar esta limitación, el constructor 210 puede construir el conjunto de datos para incluir una entrada de reubicación para cada rutina que se mapea en el conjunto de datos. Durante la generación, construcción y/o vinculación de los archivos ejecutables, las entradas de reubicación se reemplazan con la dirección de inicio real (absoluta) de las rutinas mapeadas.

Opcionalmente, el constructor 210 optimiza la disposición del conjunto de datos de entradas de rutinas ordenando las entradas en uno o más esquemas de ordenamiento ordenado, por ejemplo, de acuerdo con la dirección de inicio absoluta de las rutinas mapeadas. Las entradas ordenadas pueden aumentar significativamente la eficiencia de recorrer el conjunto de datos ordenado y, por lo tanto, reducir el tiempo de procesamiento de la búsqueda de una coincidencia con una de las direcciones mapeadas en las entradas del conjunto de datos. Dado que las direcciones de inicio absolutas de las rutinas pueden no estar disponibles antes de la etapa de generación de los archivos ejecutables, el constructor 210 puede incluir en cada una de las entradas del conjunto de datos un valor de código único que no es parte del conjunto de instrucciones de los procesadores objetivo. Después de la generación, construcción y/o vinculación de los archivos ejecutables, el constructor 210 puede buscar los códigos únicos en el archivo ejecutable, por ejemplo, un archivo ejecutable ELF para determinar la dirección de inicio absoluta de las rutinas mapeadas. El constructor 210 puede entonces actualizar el conjunto de datos y ordenar sus entradas de acuerdo con las direcciones de inicio reales de las rutinas mapeadas, por ejemplo, en un orden ascendente, en un orden descendente y/o similar.

El constructor 210 puede ajustar además el o los archivos de código intermedio ajustados para incluir el conjunto de datos que mapea la dirección de inicio de las rutinas. Adicional y/o alternativamente, el constructor 210 genera uno o más archivos de código intermedio adicionales para almacenar el conjunto de datos que mapea la dirección de inicio de las rutinas. El o los archivos de código intermedio adicionales pueden usarse durante la generación, construcción y/o vinculación del o los archivos ejecutables para incluir correctamente el conjunto de datos en la construcción final. La implementación exacta del conjunto de datos y su almacenamiento para uso en tiempo de ejecución pueden seleccionarse de acuerdo con uno o más parámetros del sistema informático al que apuntan el o los archivos ejecutables generados utilizando el o los archivos de código intermedio ajustados.

El constructor 210 puede configurar el segmento de código de verificación para comparar primero la dirección de la ramificación indicada por la instrucción de ramificación indirecta con las direcciones enumeradas en el conjunto de datos para buscar una coincidencia y determinar en consecuencia si la dirección de la ramificación es la dirección inicial de una de las rutinas mapeadas.

Durante el tiempo de ejecución, el segmento de código de verificación compara la dirección de la ramificación con las direcciones enumeradas en las entradas del conjunto de datos. En caso de que la dirección de la ramificación coincida con una de las direcciones enumeradas en el conjunto de datos, el segmento de código de verificación puede determinar que la ramificación indirecta está apuntando al inicio de una rutina válida. Sin embargo, en caso de que la dirección de la ramificación no coincida con ninguna de las direcciones enumeradas en el conjunto de datos, el segmento de código de verificación puede determinar que la ramificación indirecta no es válida ya que no está apuntando al inicio de una rutina válida. Como se describió anteriormente en la presente memoria descriptiva, el conjunto de datos puede optimizarse con sus entradas ordenadas. Por lo tanto, el segmento de código de verificación puede aplicar uno o más esquemas de búsqueda para recorrer el conjunto de datos ordenado en tiempo de ejecución en busca de una coincidencia. Esto puede mejorar significativamente el rendimiento de acceso del segmento de código de verificación y, por lo tanto, reducir significativamente el tiempo de procesamiento requerido para que el segmento de código de verificación busque la coincidencia.

Como se muestra en 310, el constructor 210 modifica datos, instrucción(es), tabla(s) de símbolos y/o uno o más atributos del(de los) archivo(s) de código intermedio afectado(s) por el ajuste del(de los) archivo(s) de código intermedio realizado para incluir el(los) segmento(s) de código de verificación como se describe en la etapa 110 del proceso 100.

Como se muestra en 312, el constructor 210 puede generar los archivos de código intermedio ajustados que pueden usarse para generar, construir y/o vincular uno o más archivos ejecutables que pueden ser ejecutados por uno o más procesadores como se describe en la etapa 112 del proceso 100.

Se espera que durante la vida de una patente que madure a partir de la presente solicitud se desarrollen muchos sistemas, procedimientos y programas informáticos relevantes y el alcance de los términos formato de archivos de código intermedio, herramientas de análisis de archivos de código intermedio están destinados a incluir todas esas nuevas tecnologías *a priori*.

Como se usa en la presente memoria descriptiva, el término “aproximadamente” se refiere a ± 10%.

Los términos “comprende”, “que comprende”, “incluye”, “que incluye”, “que tiene” y sus conjugados significan

“incluye, pero no se limita a”.

El término “que consiste en” significa “incluyendo y limitado a”.

Tal como se utiliza en la presente memoria descriptiva, la forma singular “un”, “uno”, “una”, “el” y “la” incluyen referencias en plural a menos que el contexto indique claramente lo contrario. Por ejemplo, el término “un compuesto” o “al menos un compuesto” puede incluir una pluralidad de compuestos, incluidas mezclas de los mismos.

A lo largo de la presente solicitud, se pueden presentar diversas realizaciones de la presente invención en un formato de intervalo. Debe entenderse que la descripción en formato de intervalo es meramente por conveniencia y brevedad y no debe interpretarse como una limitación inflexible del alcance de la invención. Por consiguiente, se debe considerar que la descripción de un intervalo ha divulgado específicamente todos los subintervalos posibles, así como los valores numéricos individuales dentro de ese intervalo. Por ejemplo, se debe considerar que la descripción de un intervalo como de 1 a 6 ha divulgado específicamente subintervalos como de 1 a 3, de 1 a 4, de 1 a 5, de 2 a 4, de 2 a 6, de 3 a 6, etc., así como números individuales dentro de ese intervalo, por ejemplo, 1, 2, 3, 4, 5 y 6. Esto se aplica independientemente de la amplitud del intervalo.

Siempre que se indique un intervalo numérico en la presente memoria descriptiva, se pretende que incluya cualquier numeral citado (fraccional o integral) dentro del intervalo indicado. Las frases “que oscila entre” un primer número indicado y un segundo número indicado y “que oscila de” un primer número indicado “a” un segundo número indicado se utilizan en la presente memoria descriptiva de manera intercambiable y pretenden incluir el primer y segundo números indicados y todos los números fraccionarios y enteros entre ellos.

Se aprecia que ciertas características de la invención, que, para mayor claridad, se describen en el contexto de realizaciones separadas, también se pueden proporcionar en combinación en una única realización. Por el contrario, varias características de la invención, que, para mayor brevedad, se describen en el contexto de una única realización, también se pueden proporcionar por separado o en cualquier subcombinación adecuada o según sea adecuado en cualquier otra realización descrita de la invención. Ciertas características descritas en el contexto de varias realizaciones no se deben considerar características esenciales de esas realizaciones, a menos que la realización no pueda funcionar sin esos elementos.

REIVINDICACIONES

1. Un producto de programa informático que comprende al menos un archivo ejecutable generado a partir de al menos un archivo de código intermedio ajustado para aplicar protección de dirección de retorno y/o protección de programación orientada al retorno, que comprende:

un medio de almacenamiento legible por ordenador no transitorio almacenado en el mismo: y
una pluralidad de instrucciones de programa de una pluralidad de rutinas de un archivo ejecutable generado para su ejecución por al menos un procesador a partir de al menos un archivo de código intermedio generado por un compilador, estando el al menos un archivo de código intermedio ajustado para aplicar al menos uno de: protección de dirección de retorno (100) y protección de programación orientada al retorno (300) para al menos una de la pluralidad de rutinas;
en el que el ajuste (100) del al menos un archivo de código intermedio para aplicar la protección de dirección de retorno comprende:

analizar (104) una tabla de símbolos del al menos un archivo de código intermedio para identificar la pluralidad de rutinas,
extraer (106) código binario de la al menos una rutina en función de su entrada en la tabla de símbolos,
desensamblar (106) el código binario de la al menos una rutina,
analizar (106) el código desensamblado de la al menos una rutina para identificar una instrucción de inserción en pila de dirección de retorno y al menos una correspondiente instrucción de extracción de pila de dirección de retorno en la al menos una rutina,
reemplazar (108) la instrucción de inserción en pila de dirección de retorno en la al menos una rutina con un respectivo segmento de código de prólogo y cada instrucción de extracción de pila de dirección de retorno se reemplaza con un respectivo segmento de código de epílogo, estando el respectivo segmento de código de prólogo configurado para alterar la pila después de que la dirección de retorno es insertada en la pila y estando el respectivo segmento de código de epílogo configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes ramificarse a la dirección de retorno recuperada de la pila, y
modificar (110) la tabla de símbolos para reflejar la adición del respectivo segmento de código de epílogo y el respectivo segmento de código de prólogo al menos a un archivo de código intermedio;

en el que, en tiempo de ejecución, en caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que el al menos un procesador inicie al menos una acción predefinida;
en el que el ajuste (300) del al menos un archivo de código intermedio para aplicar la protección de programación orientada al retorno comprende:

analizar (304) una tabla de símbolos del al menos un archivo de código intermedio para identificar una dirección de inicio de cada una de la pluralidad de rutinas,
agregar un código único en una dirección que precede a la dirección de inicio de cada una de la pluralidad de rutinas,
extraer (306) código binario de la al menos una rutina en función de su entrada en la tabla de símbolos,
desensamblar (306) el código binario de la al menos una rutina,
analizar (306) el código desensamblado de la al menos una rutina para identificar al menos una instrucción de ramificación indirecta en la al menos una rutina,
reemplazar (308) cada instrucción de ramificación indirecta detectada con una respectiva instrucción de ramificación directa para la invocación de un respectivo segmento de código de verificación configurado para verificar, antes de ejecutar la respectiva operación de ramificación indirecta, que la respectiva instrucción de ramificación indirecta apunta a la dirección de inicio de una de la pluralidad de rutinas,
insertar una instrucción antes de la respectiva instrucción de ramificación directa para transferir al respectivo segmento de código de verificación una dirección apuntada por la respectiva instrucción de ramificación indirecta,
configurar el respectivo segmento de código de verificación para verificar que la dirección apuntada por la función de ramificación indirecta esté precedida por el código único, y
modificar (310) la tabla de símbolos para reflejar la adición del respectivo segmento de código de verificación a dicho al menos un archivo de código intermedio;

en el que, en tiempo de ejecución, el respectivo segmento de código de verificación verifica que la respectiva instrucción de ramificación indirecta esté apuntando a la dirección de inicio de una de la pluralidad de rutinas recuperando la dirección transferida y verificando que la dirección transferida esté

precedida por el código único, en caso de que la instrucción de ramificación indirecta no esté apuntando a la dirección de inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que el al menos un procesador inicie al menos una acción predefinida;
en el que la pluralidad de instrucciones de programa son ejecutadas por el al menos un procesador desde el medio de almacenamiento legible por ordenador no transitorio.

2. El producto de programa informático según la reivindicación 1, en el que la al menos una acción predefinida es miembro de un grupo que consiste en: bloquear la ejecución del al menos un procesador, detener la ejecución del al menos un procesador, hacer que el al menos un procesador se ramifique a una dirección predefinida, impedir que el al menos un procesador ejecute al menos una instrucción de código potencialmente malicioso y generar una indicación de una alteración de pila inválida.

3. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que el al menos un archivo de código intermedio es miembro de un grupo que consiste en: un archivo de objeto, un archivo de almacenamiento y un archivo binario.

4. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que cada una de la pluralidad de rutinas es miembro de un grupo que consiste en: una rutina, una subrutina y una función.

5. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que el al menos un archivo de código intermedio se ajusta para modificar al menos uno de: una instrucción y un elemento de datos afectado por la invocación de los segmentos de prólogo, epílogo y/o código de verificación.

6. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que la tabla de símbolos se actualiza además para reflejar la invocación de los segmentos de prólogo, epílogo y/o código de verificación y un aumento del tamaño de las rutinas ajustadas.

7. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que la alteración de la pila se basa en al menos uno de: alterar la dirección de retorno de manera reversible e introducir un marcador en la pila en una ubicación adyacente a la ubicación de dirección de retorno insertada,

en el que alterar la dirección de retorno de manera reversible comprende:

configurar el respectivo segmento de código de prólogo para leer la dirección de retorno insertada en la pila, alterar la dirección de retorno de manera reversible e introducir la dirección de retorno alterada de nuevo en la pila, y
configurar el respectivo segmento de código de epílogo para leer la dirección de retorno alterada a partir de la dirección de pila, recuperar la dirección de retorno a partir de la dirección de retorno alterada invirtiendo la operación realizada por el respectivo segmento de código de prólogo e introducir la dirección de retorno recuperada de nuevo en la pila; y

en el que introducir el marcador en la pila comprende:

configurar el respectivo segmento de código de prólogo para insertar un marcador de valor constante en la ubicación adyacente en la pila, y
configurar el respectivo segmento de código de epílogo para verificar el marcador de valor constante y eliminar el marcador de valor constante de la pila.

8. El producto de programa informático según la reivindicación 7, en el que la alteración de la dirección de retorno está basada en una operación XOR de la dirección de retorno con un valor seleccionado aleatoriamente, de modo que el respectivo segmento de código de prólogo está configurado para realizar una operación XOR de la dirección de retorno con el valor seleccionado aleatoriamente y el respectivo segmento de código de epílogo está configurado para realizar una operación XOR de la dirección de retorno alterada con el mismo valor seleccionado aleatoriamente.

9. El producto de programa informático según la reivindicación 7, en el que el marcador de valor constante se selecciona aleatoriamente durante cada evento de inicio del al menos un procesador.

10. El producto de programa informático según cualquiera de las reivindicaciones precedentes, en el que el respectivo segmento de código de prólogo y cada uno de los respectivos segmentos de código de epílogo se agregan a la al menos una rutina sustituyendo la instrucción de inserción en pila de dirección de retorno y cada una de las instrucciones de extracción de pila de dirección de retorno por una función de

ramificación de trampolín configurada para invocar el respectivo segmento de código agregado durante las operaciones de inserción y extracción de dirección.

11. El producto de programa informático según la reivindicación 1, en el que ajustar la al menos una rutina comprende sustituir la instrucción de ramificación indirecta por una instrucción de ramificación directa que apunta al respectivo segmento de código de verificación.

12. El producto de programa informático según la reivindicación 11, en el que la instrucción insertada para transferir al respectivo segmento de código de verificación la dirección apuntada por la respectiva instrucción de ramificación indirecta comprende al menos uno de:

- una instrucción de inserción en pila antes de la instrucción de ramificación directa para insertar en la pila una dirección apuntada por la instrucción de ramificación indirecta, la dirección insertada después de extraerla de la pila es utilizada por el segmento de código de verificación para la verificación,
- una instrucción de escritura antes de la instrucción de ramificación directa para escribir una dirección apuntada por la instrucción de ramificación indirecta en un registro predeterminado, la dirección escrita después de ser recuperada del registro predeterminado es utilizada por el segmento de código de verificación para la verificación, y
- una instrucción de escritura antes de la instrucción de ramificación directa para escribir una dirección apuntada por la instrucción de ramificación indirecta en una variable predefinida, la dirección escrita después de ser recuperada de la variable predefinida es utilizada por el segmento de código de verificación para la verificación.

13. El producto de programa informático según cualquiera de las reivindicaciones 11 a 12, en el que la tabla de símbolos se modifica además para reflejar los segmentos de código agregados y un aumento del tamaño de la al menos una rutina ajustada.

14. Un procedimiento implementado por ordenador para ajustar archivos de código intermedio compilados para aplicar protección de dirección de retorno y/o protección de programación orientada al retorno, que comprende:

recibir (102) al menos un archivo de código intermedio generado por un compilador, comprendiendo el al menos un archivo de código intermedio una pluralidad de rutinas;
ajustar el al menos un archivo de código intermedio antes de la generación de un respectivo archivo ejecutable para su ejecución por al menos un procesador para aplicar protección de dirección de retorno (100) para al menos una de la pluralidad de rutinas mediante:

analizar (104) una tabla de símbolos del al menos un archivo de código intermedio para identificar la pluralidad de rutinas,
extraer (106) código binario de la al menos una rutina en función de su entrada en la tabla de símbolos,
desensamblar (106) el código binario de la al menos una rutina,
analizar (106) el código desensamblado de la al menos una rutina para identificar una instrucción de inserción en pila de dirección de retorno y al menos una correspondiente instrucción de extracción de pila de dirección de retorno en al menos una de la pluralidad de rutinas,
reemplazar (108) la instrucción de inserción en pila de dirección de retorno detectada en al menos una de la pluralidad de rutinas con un respectivo segmento de código de prólogo y reemplazar cada instrucción de extracción de pila de dirección de retorno detectada en la al menos una rutina con un respectivo segmento de código de epílogo para aplicar la protección de dirección de retorno, estando el respectivo segmento de código de prólogo configurado para alterar la pila después de que la dirección de retorno es insertada en la pila y estando el respectivo segmento de código de epílogo configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes ramificarse a la dirección de retorno recuperada de la pila, y
modificar (110) la tabla de símbolos para reflejar la adición del respectivo segmento de código de epílogo y el respectivo segmento de código de prólogo al menos a un archivo de código intermedio; y/o

ajustar el al menos un archivo de código intermedio antes de la generación de un respectivo archivo ejecutable para su ejecución por al menos un procesador para aplicar protección de programación orientada al retorno (300) para al menos una de la pluralidad de rutinas mediante:

analizar (304) una tabla de símbolos del al menos un archivo de código intermedio para identificar una dirección de inicio de cada rutina,
agregar un código único en una dirección que precede a la dirección de inicio de cada una de la

pluralidad de rutinas,
extraer (304) código binario de la al menos una rutina de acuerdo con su entrada en la tabla de símbolos,
desensamblar (304) el código binario de la al menos una rutina,
analizar (306) el código desensamblado de la al menos una rutina para identificar al menos una instrucción de ramificación indirecta en al menos una de la pluralidad de rutinas,
reemplazar (308) cada instrucción de ramificación indirecta detectada en al menos una de la pluralidad de rutinas con una respectiva instrucción de ramificación directa para la invocación de un respectivo segmento de código de verificación configurado para verificar, antes de ejecutar la respectiva operación de ramificación indirecta, que la respectiva instrucción de ramificación indirecta apunta a la dirección de inicio de una de la pluralidad de rutinas,
insertar una instrucción antes de la respectiva instrucción de ramificación directa para transferir al respectivo segmento de código de verificación una dirección apuntada por la respectiva instrucción de ramificación indirecta,
configurar el respectivo segmento de código de verificación para verificar que la dirección apuntada por la función de ramificación indirecta esté precedida por el código único, y
modificar (310) la tabla de símbolos para reflejar la adición del respectivo segmento de código de verificación a dicho al menos un archivo de código intermedio; y

emitir (112) el al menos a un archivo de código intermedio ajustado;
en el que, en tiempo de ejecución, en caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que el al menos a un procesador inicie al menos una acción predefinida;
en el que, en tiempo de ejecución, el respectivo segmento de código de verificación verifica que la respectiva instrucción de ramificación indirecta esté apuntando a la dirección de inicio de una de la pluralidad de rutinas recuperando la dirección transferida y verificando que la dirección transferida esté precedida por el código único, en caso de que la instrucción de ramificación indirecta no esté apuntando a la dirección de inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que el al menos un procesador inicie al menos una acción predefinida.

15. Un sistema para ajustar archivos de código intermedio compilados para aplicar protección de dirección de retorno y/o protección de programación orientada al retorno, que comprende:

un almacenamiento de programas (206) que almacena un código (210); y
al menos un procesador (204) acoplado al almacenamiento de programas (206) para ejecutar el código almacenado (210), comprendiendo el código (210):

instrucciones de código para recibir (102) al menos un archivo de código intermedio generado por un compilador, comprendiendo el al menos un archivo de código intermedio una pluralidad de rutinas;
instrucciones de código para ajustar el al menos un archivo de código intermedio antes de la generación de un respectivo archivo ejecutable para su ejecución por al menos un procesador para aplicar protección de dirección de retorno (100) para al menos una de la pluralidad de rutinas mediante:

analizar (104) una tabla de símbolos del al menos un archivo de código intermedio para identificar cada una de la pluralidad de rutinas,
extraer (106) código binario de la al menos una rutina en función de su entrada en la tabla de símbolos,
desensamblar (106) el código binario de la al menos una rutina,
analizar (106) el código desensamblado de la al menos una rutina para identificar una instrucción de inserción en pila de dirección de retorno y al menos una correspondiente instrucción de extracción de pila de dirección de retorno,
reemplazar (108) la instrucción de inserción en pila de dirección de retorno detectada en al menos una de la pluralidad de rutinas con un respectivo segmento de código de prólogo y reemplazar cada instrucción de extracción de pila de dirección detectada en la al menos una rutina con un respectivo segmento de código de epílogo para aplicar la protección de dirección de retorno, estando el respectivo segmento de código de prólogo configurado para alterar la pila después de que la dirección de retorno es insertada en la pila y estando el respectivo segmento de código de epílogo configurado para validar la alteración de la pila realizada por el respectivo segmento de código de prólogo antes ramificarse a la dirección de retorno recuperada de la pila, y
modificar (110) la tabla de símbolos para reflejar la adición del respectivo segmento de código de epílogo y el respectivo segmento de código de prólogo al menos a un archivo de código intermedio; y/o

instrucciones de código para ajustar el al menos un archivo de código intermedio antes de la generación de un respectivo archivo ejecutable para su ejecución por al menos un procesador para aplicar protección de programación orientada al retorno (300) para al menos una de la pluralidad de rutinas mediante:

analizar (104) una tabla de símbolos del al menos un archivo de código intermedio para identificar una dirección de inicio de cada rutina,
agregar un código único en una dirección que precede a la dirección de inicio de cada una de la pluralidad de rutinas,
extraer (304) código binario de la al menos una rutina de acuerdo con su entrada en la tabla de símbolos,
desensamblar (304) el código binario de la al menos una rutina,
analizar (306) el código desensamblado de la al menos una rutina para identificar al menos una instrucción de ramificación indirecta en al menos una de la pluralidad de rutinas,
reemplazar (308) cada instrucción de ramificación indirecta detectada en la al menos una rutina con una respectiva instrucción de ramificación directa para la invocación de un respectivo segmento de código de verificación configurado para verificar, antes de ejecutar la respectiva operación de ramificación indirecta, que la respectiva instrucción de ramificación indirecta apunta a la dirección de inicio de una de la pluralidad de rutinas para aplicar la protección de programación orientada al retorno,
insertar una instrucción antes de la respectiva instrucción de ramificación directa para transferir al respectivo segmento de código de verificación una dirección apuntada por la respectiva instrucción de ramificación indirecta,
configurar el respectivo segmento de código de verificación para verificar que la dirección apuntada por la función de ramificación indirecta esté precedida por el código único, y
modificar (310) la tabla de símbolos para reflejar la adición del respectivo segmento de código de verificación a dicho al menos un archivo de código intermedio; e

instrucciones de código para emitir (112) el al menos un archivo de código intermedio ajustado;

en el que, en tiempo de ejecución, en caso de que no se pueda validar la alteración de la pila, el respectivo segmento de código de epílogo hace que el al menos un procesador inicie al menos una acción predefinida;

en el que, en tiempo de ejecución, el respectivo segmento de código de verificación verifica que la respectiva instrucción de ramificación indirecta esté apuntando a la dirección de inicio de una de la pluralidad de rutinas recuperando la dirección transferida y verificando que la dirección transferida esté precedida por el código único, en caso de que la instrucción de ramificación indirecta no esté apuntando a la dirección de inicio de una de la pluralidad de rutinas, el respectivo segmento de código de verificación hace que el al menos un procesador inicie al menos una acción predefinida.

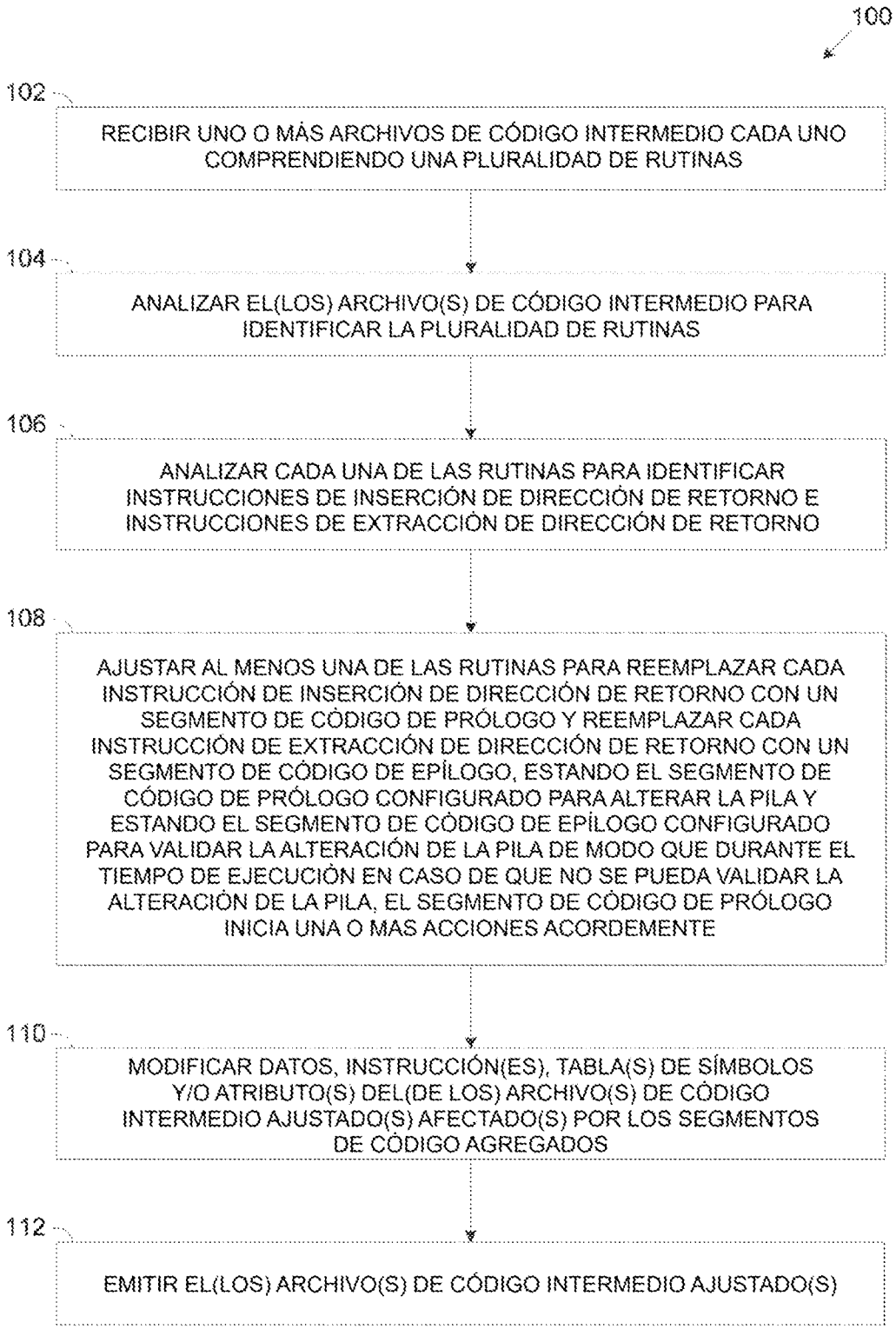
100
102
RECIBIR UNO O MÁS ARCHIVOS DE CÓDIGO INTERMEDIO CADA UNO COMPRENDIENDO UNA PLURALIDAD DE RUTINAS
104
ANALIZAR EL(LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO PARA IDENTIFICAR LA PLURALIDAD DE RUTINAS
106
ANALIZAR CADA UNA DE LAS RUTINAS PARA IDENTIFICAR INSTRUCCIONES DE INSERCIÓN DE DIRECCIÓN DE RETORNO E INSTRUCCIONES DE EXTRACCIÓN DE DIRECCIÓN DE RETORNO
108
AJUSTAR AL MENOS UNA DE LAS RUTINAS PARA REEMPLAZAR CADA INSTRUCCIÓN DE INSERCIÓN DE DIRECCIÓN DE RETORNO CON UN SEGMENTO DE CÓDIGO DE PRÓLOGO Y REEMPLAZAR CADA INSTRUCCIÓN DE EXTRACCIÓN DE DIRECCIÓN DE RETORNO CON UN SEGMENTO DE CÓDIGO DE EPÍLOGO, ESTANDO EL SEGMENTO DE CÓDIGO DE PRÓLOGO CONFIGURADO PARA ALTERAR LA PILA Y ESTANDO EL SEGMENTO DE CÓDIGO DE EPÍLOGO CONFIGURADO PARA VALIDAR LA ALTERACIÓN DE LA PILA DE MODO QUE DURANTE EL TIEMPO DE EJECUCIÓN EN CASO DE QUE NO SE PUEDA VALIDAR LA ALTERACIÓN DE LA PILA, EL SEGMENTO DE CÓDIGO DE PRÓLOGO INICIA UNA O MAS ACCIONES ACORDEMENTE
110
MODIFICAR DATOS, INSTRUCCIÓN(ES), TABLA(S) DE SÍMBOLOS Y/O ATRIBUTO(S) DEL(DE LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO AJUSTADO(S) AFECTADO(S) POR LOS SEGMENTOS DE CÓDIGO AGREGADOS
112
EMITIR EL(LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO AJUSTADO(S)

FIG. 1

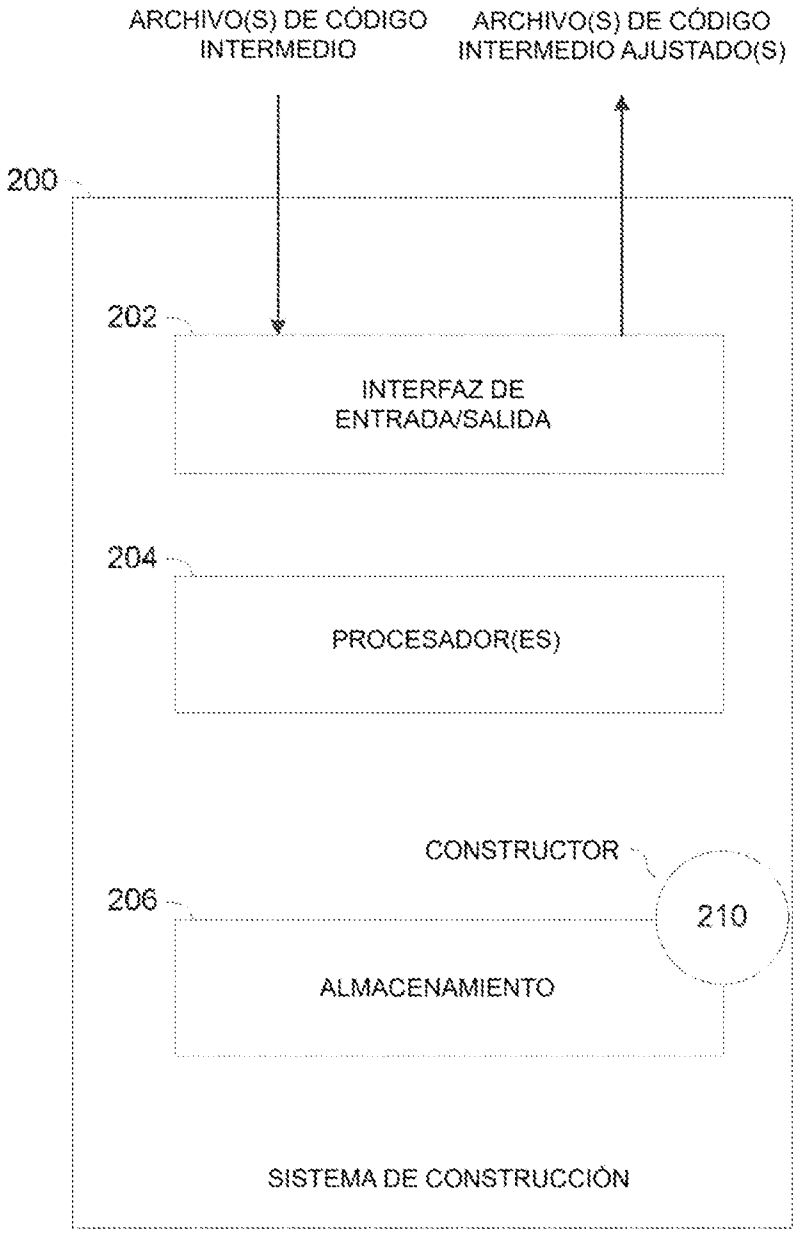
ARCHIVO(S) DE CÓDIGO INTERMEDIO
ARCHIVO(S) DE CÓDIGO INTERMEDIO AJUSTADO(S)
200
202
INTERFAZ DE ENTRADA/SALIDA
204
PROCESADOR(ES)
CONSTRUCTOR
210
206
ALMACENAMIENTO
SISTEMA DE CONSTRUCCIÓN

FIG. 2

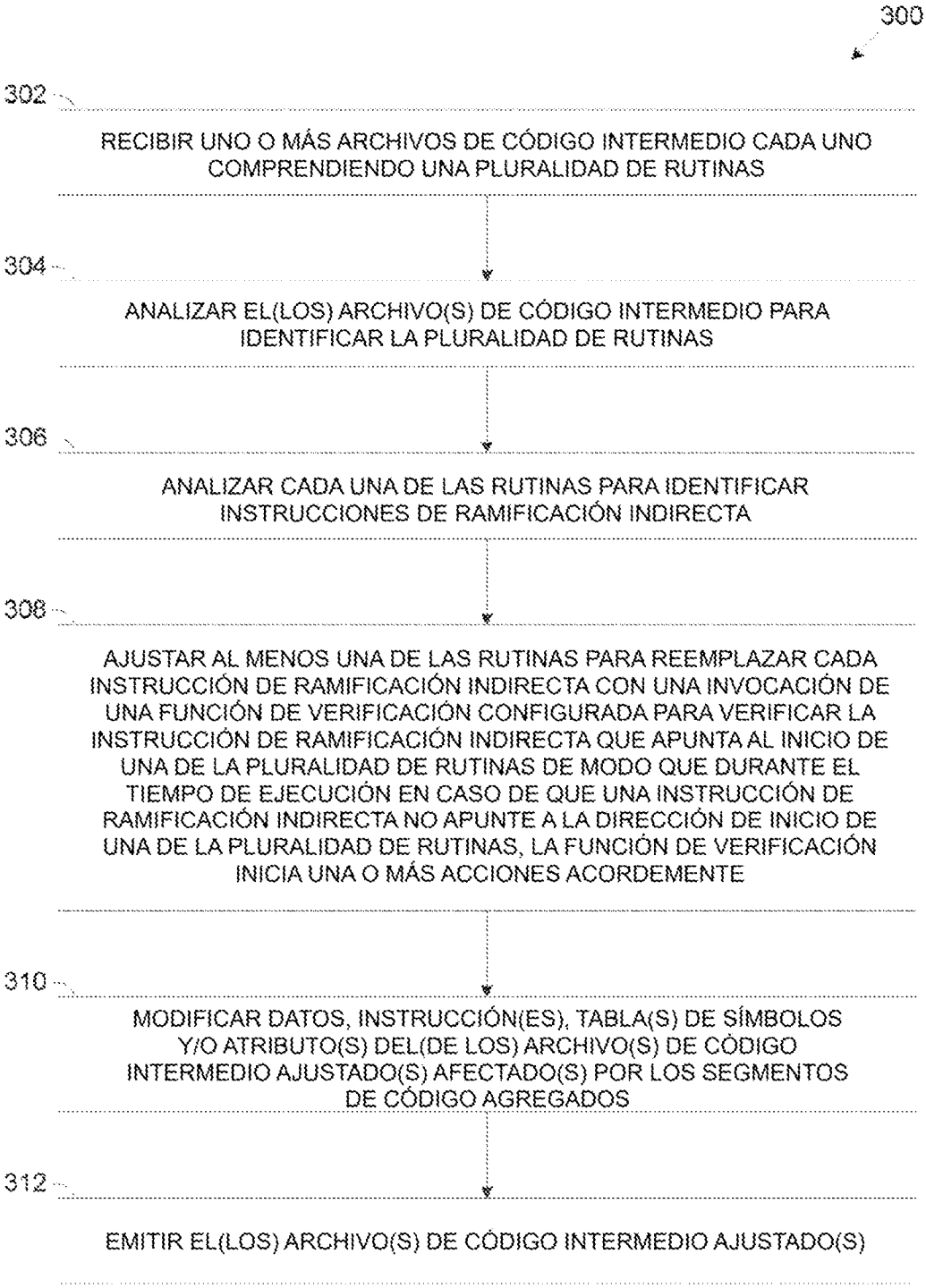
300
302
RECIBIR UNO O MÁS ARCHIVOS DE CÓDIGO INTERMEDIO CADA UNO COMPRENDIENDO UNA PLURALIDAD DE RUTINAS
304
ANALIZAR EL(LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO PARA IDENTIFICAR LA PLURALIDAD DE RUTINAS
306
ANALIZAR CADA UNA DE LAS RUTINAS PARA IDENTIFICAR INSTRUCCIONES DE RAMIFICACIÓN INDIRECTA
308
AJUSTAR AL MENOS UNA DE LAS RUTINAS PARA REEMPLAZAR CADA INSTRUCCIÓN DE RAMIFICACIÓN INDIRECTA CON UNA INVOCACIÓN DE UNA FUNCIÓN DE VERIFICACIÓN CONFIGURADA PARA VERIFICAR LA INSTRUCCIÓN DE RAMIFICACIÓN INDIRECTA QUE APUNTA AL INICIO DE UNA DE LA PLURALIDAD DE RUTINAS DE MODO QUE DURANTE EL TIEMPO DE EJECUCIÓN EN CASO DE QUE UNA INSTRUCCIÓN DE RAMIFICACIÓN INDIRECTA NO APUNTE A LA DIRECCIÓN DE INICIO DE UNA DE LA PLURALIDAD DE RUTINAS, LA FUNCIÓN DE VERIFICACIÓN INICIA UNA O MÁS ACCIONES ACORDEMENTE
310
MODIFICAR DATOS, INSTRUCCIÓN(ES), TABLA(S) DE SÍMBOLOS Y/O ATRIBUTO(S) DEL(DE LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO AJUSTADO(S) AFECTADO(S) POR LOS SEGMENTOS DE CÓDIGO AGREGADOS
312
EMITIR EL(LOS) ARCHIVO(S) DE CÓDIGO INTERMEDIO AJUSTADO(S)

FIG. 3