(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 302 962**

(51) Int. Cl.:
G06F 21/00 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Número de solicitud europea: **03780354 .1**
(86) Fecha de presentación : **08.12.2003**
(87) Número de publicación de la solicitud: **1573465**
(87) Fecha de publicación de la solicitud: **14.09.2005**

(54) Título: **Método y sistema para detectar de forma heurística virus en un código ejecutable.**

(30) Prioridad: **12.12.2002 GB 0229032**

(45) Fecha de publicación de la mención BOPI:
01.08.2008

(45) Fecha de la publicación del folleto de la patente:
01.08.2008

(73) Titular/es: **MessageLabs Limited**
1240 Lansdowne Court Gloucester Business Park
Gloucester, Gloucestershire GL3 4AB, GB

(72) Inventor/es: **Shipp, Alexander**

(74) Agente: **Elzaburu Márquez, Alberto**

DESCRIPCIÓN

Método y sistema para detectar de forma heurística virus en un código ejecutable.

El presente invento se refiere a un método y un sistema para detectar de forma heurística virus en código ejecutable mediante el análisis de la distribución de frecuencias del código máquina creado.

Una forma común de infección por virus informáticos es aquella en la que el código ejecutable del virus está unido a, o embebido en, un programa u otro fichero informático que contiene código ejecutable, el cual, a primera vista, parece ser benigno. Un método establecido de propagación de virus es aquel en el que el virus, una vez activado en una máquina anfitriona tal como un PC de usuario, se unirá a uno o más programas que encuentre en dicha máquina anfitriona de tal manera que ese programa, cuando se ejecute, ejecutará el código del virus dándole la oportunidad de propagarse otra vez y/o de emprender cualesquiera otros comportamientos malignos (tales como destrucción de ficheros, etc.) que hayan sido programados en él. Este método de propagación proporciona, por supuesto, una oportunidad de detectar el virus, por ejemplo mediante la asociación de sumas de comprobación a ficheros de programa y la detección de cuándo esta suma de comprobación cambia. Esa es, por supuesto, sólo una de las muchas estrategias que se han ideado para detectar virus.

Otro método muy conocido para detectar virus, implementado en muchos de los paquetes de software anti-virus que están disponibles, implica explorar el programa y otros ficheros en busca de ciertas secuencias características de bytes (conocidas como firmas), las cuales indican la presencia probable de un virus. Uno de los problemas prácticos que existen con la detección basada en firmas es que se requiere algo de habilidad y una cantidad considerable de tiempo, cuando se detecta por primera vez un nuevo virus, para establecer una firma característica apropiada de dicho virus. Esta firma debe ser una que no produzca demasiados positivos falsos y que no identifique erróneamente al virus, por ejemplo como uno ya existente que tenga una carga más benigna. A continuación, esta información de la firma debe ser difundida a los sitios que usen el paquete anti-virus en cuestión antes de que pueda ser utilizada en ellos para detectar los virus recién identificados. En los últimos años, muchos de los ataques notables por virus han implicado a virus que se propagan por la red de Internet y a los publicadores de software anti-virus les lleva tiempo reaccionar cuando se produce un ataque de un virus.

Algunos proveedores de servicios de Internet (ISP) ofrecen exploración anti-virus del tráfico de Internet que pasa a través de sus nodos de Internet como un servicio de valor añadido. El documento WO 01/69356 explica un método para detectar virus de forma heurística utilizando un histograma.

El presente invento se refiere a un método de detección de virus que tiene la finalidad de ser útil para ISPs que realicen exploración anti-virus, por ejemplo, de ejecutables tales como ficheros de programa adjuntos a correos electrónicos, aunque de ningún modo está limitado a esa aplicación y se puede utilizar en cualquier paquete anti-virus.

De acuerdo con el presente invento se proporciona un método de exploración de un fichero informático en busca de infecciones por virus que comprende:

a) identificar el código de programa del fichero;

b) identificar el compilador utilizado para crear el código de programa;

c) determinar la distribución de frecuencias de instrucciones seleccionadas de código máquina o de secuencias de dichas instrucciones en el código de programa; y

d) marcar el fichero como posiblemente infectado por un virus, o no, sobre la base de la comparación de la distribución de frecuencias determinada con una distribución de frecuencias de instrucciones de código máquina o de secuencias de dichas instrucciones esperada para ese compilador.

El invento también proporciona un sistema para explorar un fichero informático en busca de infecciones por virus que comprende:

a) medios para identificar el código de programa del fichero;

b) medios para identificar el compilador utilizado para crear el código de programa;

c) medios para determinar la distribución de frecuencias de instrucciones seleccionadas de código máquina o secuencias de dichas instrucciones en el código de programa; y

d) medios para marcar el fichero como posiblemente infectado por un virus, o no, sobre la base de la comparación de la distribución de frecuencias determinada con una distribución de frecuencias de instrucciones de código máquina o de secuencias de dichas instrucciones esperada para ese compilador.

Se describirá el invento en mayor detalle a modo de ejemplo no limitativo haciendo referencia a los dibujos adjuntos, en los cuales:

La Figura 1 es una combinación de un diagrama de bloques y un diagrama de flujo del funcionamiento de un motor de búsqueda de virus de acuerdo con una realización del presente invento; y

La Figura 2 es un diagrama de flujo del funcionamiento de un ejemplo del analizador de frecuencias de instrucciones de la Figura 1.

En lo que sigue, los valores hexadecimales se representan con un 0x delante, como en este caso: 0xff78. La expresión 0x???? se utiliza para representar un valor hexadecimal cuando el valor no es importante.

En primer lugar, se presentará algo de terminología.

"Suma MD5 de comprobación (resumen de mensajes 5)": MD5 es un algoritmo hash unidireccional - genera un número grande (la suma MD5 de comprobación) después de analizar una secuencia de bytes - tal como un fichero. Las posibilidades de que dos ficheros generen el mismo número grande son muy pequeñas. Es también muy difícil crear un fichero que genere cualquier suma MD5 de comprobación concreta.

"Positivo falso": Se produce un positivo falso cuando un producto anti-virus identifica a un fichero "a" concreto como de tipo malintencionado, cuando en la realidad no lo es.

"Expresión regular": Las expresiones regulares son cadenas de texto ordenadas que se pueden utilizar para reconocimiento de patrones. Por ejemplo, la expresión regular

/^hello [0-9]+/

coincide con cualquier cadena de texto ordenada que comience con las letras "hello", siga con un espacio, y lleve después uno o más dígitos.

"Mapa de memoria": Un mapa de memoria es una correspondencia una a una de las posiciones que ocuparía un programa cuando se carga en memoria, con algunas otras posiciones. De esta manera, si un programa ocupase las posiciones 0x400000 a 0x410000 cuando es cargado, podríamos construir un mapa de memoria desde 0x100000 a 0x110000. Siempre que el programa se refiera a una posición concreta, determinaríamos (en este caso) la posición equivalente en nuestro mapa de memoria restando 0x300000. De esta forma, 0x400000 corresponde a 0x100000, 0x400001 corresponde a 0x100001, y así sucesivamente.

"Compilador": De acuerdo con su uso estricto, un compilador genera uno o más módulos objeto a partir de código fuente de programa. Estos módulos objeto no son típicamente programas ejecutables por sí mismos, sino que requieren un paso adicional de enlazado mediante un programa enlazador. La acción de un programa enlazador es típicamente generar una imagen de un ejecutable al enlazar entre sí el módulo o los módulos objeto con bibliotecas binarias externas a las cuales hacen referencia el módulo o módulos; la producción de la imagen puede implicar el preanexado de una región de cabecera de acuerdo con un diseño de fichero ejecutable de un sistema operativo objetivo así como la adición de recursos tales como mapas de bits y similares. El término "compilador" tal como se usa en este texto, tiene la finalidad de incluir un programa enlazador, si es necesario desde un punto de vista técnico. Lo que produce el compilador no es necesariamente un programa autónomo, por supuesto: los compiladores también producen ejecutables tales como bibliotecas de enlace dinámico y controladores para dispositivos.

Los compiladores tienen a menudo indicadores del compilador (también conocidos como "interruptores"), los cuales pueden ser fijados por el usuario e influyen en el proceso de compilación y en el código generado. Por ejemplo, los indicadores del compilador pueden controlar si el código generado está optimizado en cuanto a velocidad, tamaño del código, o ninguno de los dos, si los marcos de las pilas se utilizan para llamadas a subrutinas, etc. Diferentes configuraciones de estos indicadores pueden influir en la distribución de frecuencias de instrucciones en el código generado, y algunas realizaciones del invento pueden reflejar esto teniendo datos de frecuencia esperada para una variedad de combinaciones de configuraciones de indicador del compilador por compilador.

El objetivo es que el término "fichero informático", tal como se usa en este texto, se entienda en un sentido general y, en concreto, que no esté restringido a ficheros residentes en disco.

Para conseguir el control, un virus se debe insertar a sí mismo en el interior de la ruta de ejecución del código de programa. El código del virus habrá sido creado originariamente por un compilador o ensamblador concreto y, en general, se insertará a sí mismo en el interior de un programa creado por un compilador o ensamblador diferente. A menudo, un compilador concreto generará código que podrá ser reconocido como procedente de ese compilador o familia de compiladores. Si es éste el caso, puede ser entonces posible determinar que el código viral insertado no ha sido generado por el compilador que generó el resto del programa, comparando la distribución real de frecuencias de instrucciones del programa con la distribución esperada de frecuencias de instrucciones generadas por el compilador identificado. A continuación se puede marcar el programa como sospechoso o como infectado por un virus.

La Figura 1 muestra en forma de bloques, una forma de sistema 10 de detección de virus que incluye al presente invento, el cual se puede incorporar al interior de un motor de búsqueda de virus. El funcionamiento global de este sistema 10 es tal como se describe a continuación:

Los ficheros que deben ser explorados son suministrados de forma sucesiva a una entrada 20, por ejemplo, desde una cola de entrada; cómo se colocan los ficheros dentro de esta cola y desde qué fuente(s) no son cosas directamente relevantes para el presente invento, pero podrían ser, por ejemplo, ficheros adjuntos a correos electrónicos que estén siendo procesados por un portal de correo electrónico en un ISP, o ficheros en un directorio que esté siendo procesado mediante una operación de exploración de disco.

Cada fichero que debe ser procesado se hace pasar a un analizador 30 del tipo de fichero que intenta identificar el tipo del fichero a partir de sus contenidos. Por ejemplo, puede ser o no un programa. Un fichero que no es un programa no se sigue analizando y el procesamiento se interrumpe en 40. Un fichero que se considera un programa se clasifica además en función de su tipo - por ejemplo, DOS, Windows PE, Windows NE, Linux ELF, Macintosh, etc. Si el analizador 30 del tipo de fichero determina que el tipo de fichero es conocido, el fichero es procesado a continuación por el analizador 50 del compilador, el cual intenta identificar el compilador utilizado para generar el código del fichero; si fracasa en esto, el procesamiento del fichero se interrumpe en 40, en caso contrario el fichero es procesado a continuación por un analizador 60 de frecuencias de instrucciones.

El analizador 60 realiza la descompilación del programa de forma eficaz y prepara una tabulación de la distribución de frecuencias de ciertos códigos de operaciones y/o construcciones de códigos de operaciones del código máquina, tal como se describirá con mayor detalle más adelante. Esta tabulación se pasa a un comprobador 70 de distribución de frecuencias donde se compara con uno o más conjuntos de distribuciones de frecuencias características para el compilador identificado que se encuentran almacenados en una base de datos 80. Cualquier compilador/enlazador dado puede ser capaz de generar más de un tipo de ejecutable (aplicación GUI, aplicación de consola, controlador de dispositivo, etc.) y el ciclo de compilación/enlazado se puede ver afectado por la configuración de uno o más indicadores del compilador/enlazador (por ejemplo, indicadores para controlar la creación o no de marcos de las pilas para llamadas a subrutinas, para indicar si el programa generado es una versión depurada, etc.), los cuales pueden producir diferentes distribuciones de frecuencias esperadas que se pueden almacenar en la base de datos y seleccionarse individualmente para que sean analizadas por el comprobador 70 de distribución de frecuencias.

Si el comprobador 70 de distribución de frecuencias determina que la distribución de frecuencias real procedente del analizador 60 coincide de forma suficiente exacta con la esperada para el compilador identificado, el procesamiento del fichero se interrumpe en 40; en caso contrario se considera que el fichero es sospechoso y que contiene potencialmente un virus. Para reducir el número de positivos falsos, los ficheros sospechosos son analizados por el comprobador de la lista de excepciones comparándolos con una lista de excepciones, es decir, ficheros que aunque son sospechosos según el comprobador 70 de distribución de frecuencias, se pueden considerar sin embargo benignos. El comprobador de la lista de excepciones puede operar mediante referencia a una lista de excepciones almacenada en la base de datos 80 junto con características utilizadas para determinar si el fichero que se está analizando coincide con una excepción. Si el fichero no coincide con una excepción, es marcado como viral en la salida 100. La configuración de este indicador se puede utilizar para alertar a un operador, y/o para iniciar el procesamiento adicional del fichero y/o para iniciar una acción correctiva apropiada (por ejemplo, poner el fichero en cuarentena).

Reconocer un ejecutable

Lo que sigue es un ejemplo simplista de un algoritmo para determinar si es probable que un fichero sea un ejecutable que podría ser utilizado por el analizador 30 del tipo de fichero. Analizando los primeros pocos bytes de un fichero es posible saber si es probable que sea un ejecutable. Por ejemplo, para reconocer un fichero de Windows PE:

Leer los primeros 2 bytes. Si éstos no son “MZ” entonces parar.

Leer otros 58 bytes.

Leer 4 bytes y cargarlos sobre la variable x (tratar utilizando el ordenamiento de bytes de Intel)

Situarse sobre el desplazamiento x en el fichero.

Leer 4 bytes

Si los bytes son P E \0 \0, entonces es probable que el fichero sea un fichero de Windows PE.

Este algoritmo se puede mejorar para añadir reconocimiento de tantos otros tipos de fichero ejecutable como se desee. Por ejemplo, si los primeros 4 bytes de un fichero son 0x7F 0x45 0x46, entonces es probable que el fichero sea un ejecutable de Linux que utilice el formato ELF.

Reconociendo el compilador

Existen diferentes formas mediante las cuales el analizador 50 del compilador puede reconocer qué compilador creó un programa concreto. Por ejemplo, podría examinar la secuencia de arranque del programa, o las secuencias de llamada y retorno de subrutinas. En algunos casos, esto es suficiente para identificar la versión exacta de compilador utilizada. En otros, esto identificará una posible familia de compiladores.

Descompilación del programa

Lo siguiente es un método simplista mediante el cual esto puede ser llevado a cabo por el analizador 60 de frecuencias de instrucciones. Este método se ilustra en la Figura 2 y es como sigue:

Crear un mapa de memoria de las posiciones utilizadas por el programa, marcando cada byte como "no usado" (paso 210).

Empujar el punto de entrada del programa sobre una pila de posiciones a considerar (220).

Mientras hay todavía posiciones a considerar (230)

Coger la siguiente posición como "posición actual" (240)

LblNext:

Si el mapa de memoria marca este byte como "código" (250), detener el procesamiento de esta posición

Leer la instrucción en esta posición, calculando su longitud en bytes (260)

Actualizar el contador de frecuencias para esta instrucción (270)

Marcar "longitud" bytes en el mapa de memoria como "código" (280)

Si la instrucción es un "call", "jump" u otra instrucción que pudiera cambiar la posición de la siguiente instrucción (290), empujar el destino sobre la pila de posiciones a considerar (300)

Si la instrucción es una instrucción del tipo "jump always" o "return", detener el procesamiento de esta posición (310)

Si la instrucción carga o almacena datos en posiciones concretas (320), marcar el destino en el mapa de memoria como "datos posibles" (330)

Incrementar "posición actual" en un número "longitud" de bytes (340)

Continuar procesamiento en lblnext

Wend

Este algoritmo se puede mejorar de muchas maneras para que proporcione mejores resultados. Por ejemplo, una vez que ha terminado el procesamiento, el mapa de memoria tendrá áreas marcadas como "código", "datos posibles" y "no usado". Si hay demasiadas áreas marcadas como "no usado" entonces se puede emprender un análisis adicional de estas áreas para intentar determinar si son código o datos. Un algoritmo de este tipo podría ser comprobar datos para ver si contienen caracteres en el rango 0x20 a 0x7F, más también 0x0A, 0x09 0x0d, finalizando con 0x00 o "$". Si es así, éste podría ser un mensaje mostrado por el programa que se está analizando y se puede marcar como datos. También se pueden analizar los bytes inmediatamente anteriores al mensaje para ver si parecen tener una longitud del mensaje de 1, 2 o 4 bytes. Son posibles muchos otros algoritmos. Ciertos tipos de ficheros de programa, por ejemplo las bibliotecas de enlace dinámico de Windows, pueden contener múltiples puntos de entrada y los algoritmos anteriores se pueden aplicar a cada uno de ellos.

Comprobación de frecuencias conocidas

Compiladores concretos harán la misma cosa de la misma manera en cada momento (cuando se use el mismo conjunto de indicadores del compilador). Por ejemplo, si el compilador "A" quiere añadir uno al registro EAX, puede generar el siguiente código:

```
add     eax, 0x01
```

El registro eax tiene una longitud de 4 bytes. Sin embargo, el compilador genera una instrucción para añadir un valor de un byte, sabiendo que el procesador rellenará correctamente desde 0x01 hasta 0x00000001.

Sin embargo, ésta no es la única manera de añadir uno al registro EAX, así que si encontramos cualquiera de los siguientes códigos en un programa generado por el compilador A, esto sería sospechoso:

```
inc     eax
add     eax, 0x0001             # valor de dos bytes usado
add     eax, 0x00000001         # valor de cuatro bytes usado
```

Muchos compiladores generan secuencias de entrada y salida concretas para subrutinas.

Supongamos que el compilador genera siempre lo siguiente:

Rutina

```
#aquí está el código de entrada
push    ebp
mov     ebp, esp
sub     esp, 0x????

 ...

#aquí está el código de salida
mov     esp, ebp
pop     ebp
retn
```

Entonces si el programa contiene 100 instrucciones “retn”, y si sólo genera instrucciones retn durante la secuencia de salida de la subrutina, esperaríamos ver también al menos 100 instrucciones “push ebp”, “mov ebp, esp”, “sub esp 0x????”, “mov esp, ebp” y “pop ebp”. Cualquier cosa menor que esto indicaría que se ha introducido un código posiblemente viral.

El compilador puede también tener una forma concreta de llamar a las subrutinas:

```
call    0x????
add     esp, 0x????
```

De esta manera, si el programa contiene 100 instrucciones “call”, esperaríamos ver al menos 100 instrucciones “add esp, 0x????”.

El compilador puede no generar nunca instrucciones concretas. De esta manera, si el programa contiene una o más de éstas, esto indicaría que se ha introducido código posiblemente viral.

Por ejemplo int 3, que en los procesadores de la serie x86 de Intel es una instrucción de interrupción de depuración

Reglas de excepción

Se pueden añadir diferentes reglas de excepción a la base de datos 80 y pueden ser aplicadas por el comprobador 90 de la lista de excepciones. Como ejemplo, las instrucciones int 3 son comunes en virus, pero también pueden estar presentes en versiones depuradas de programas. De esta manera, una regla podría ser que se ignore la presencia de instrucciones “int 3” si se determina que el programa es una versión depurada.

Otras instrucciones son utilizadas por programas de sistema o programas kernel, pero no por programas de usuario. De esta forma, si esas instrucciones están presentes, pueden ser ignoradas si se determina que un programa es un programa de sistema o kernel.

Los programas compilados con un compilador pueden ser enlazados con código procedente de bibliotecas creadas por otros compiladores. Estas bibliotecas se pueden detectar mediante reconocimiento de patrones y expresiones regulares, y excluirse del análisis. Este paso se podría realizar también antes del paso 3 (descompilación) para marcar áreas como "excluir del análisis".

Algunos ficheros ejecutables concretos se pueden excluir comparando una suma md5 de comprobación del programa con una lista de md5s de exclusión.

Mejoras

Además de utilizar esto como un algoritmo autónomo de detección de virus, se puede combinar con otras técnicas como parte de un sistema mayor. Por ejemplo, a los programas marcados como sospechosos por este método se les puede asignar una cierta puntuación, o una variedad de puntuaciones dependiendo de qué ensayos superen y cuáles no. Las puntuaciones también se pueden asignar utilizando otras técnicas heurísticas, y sólo si la puntuación total sobrepasa algún límite se marca el programa como viral.

El sistema también se puede utilizar como un indicador de qué partes del programa hay que analizar en más detalle. Por ejemplo, si se han encontrado distribuciones inusuales, se puede volver a analizar el programa para descubrir dónde aparecen éstas, y se pueden determinar los límites de "código extraño". Este código marcado puede ser sometido a continuación a un análisis detallado para intentar determinar lo que el código está haciendo realmente. Si está borrando ficheros o enviando correos electrónicos masivos, por ejemplo, entonces eso es una indicación probable de que el programa es viral.

REIVINDICACIONES

1. Un método para explorar un fichero informático en busca de infecciones por virus que comprende:

a) identificar el código de programa del fichero;

b) identificar el compilador utilizado para crear el código de programa;

c) determinar la distribución de frecuencias de instrucciones seleccionadas de código máquina seleccionadas o de secuencias de dichas instrucciones en el código de programa; y

d) marcar el fichero como posiblemente infectado por un virus, o no, sobre la base de la comparación de la distribución de frecuencias determinada con una distribución de frecuencias de instrucciones de código máquina o de secuencias de las mismas esperada para ese compilador.

2. Un método de acuerdo con la reivindicación 1, en el cual el paso c) comprende el paso, trabajando desde un punto de entrada del programa, de

b1) trazar un gráfico de ejecución mediante el decodificado de códigos de operación de instrucción sucesivos y actualizar los contadores de frecuencias de las instrucciones decodificadas según avanza este trazado.

3. Un método de acuerdo con la reivindicación 2, en el cual cuando, durante el paso b1), se encuentra una llamada a subrutina o una instrucción de bifurcación condicional, el destino de la llamada o instrucción de bifurcación es empujado sobre una pila, el trazado avanza hacia el interior de la llamada a subrutina, y cuando se encuentra una instrucción de retorno, se hace salir la posición empujada de la pila y el trazado continúa con las siguientes instrucciones, si existe alguna.

4. Un método de acuerdo con la reivindicación 1, 2 ó 3, en el cual el código de programa se examina en busca de construcciones de código de operación, tales como llamada a subrutina y retorno de subrutina, secuencias de instrucción que se espera que aparezcan en una proporción determinada entre sí y, si la proporción encontrada en la realidad se diferencia de la conocida en más de una cierta cantidad, se marca el fichero como posiblemente viral, o se le somete a procesamiento adicional.

5. Un método de acuerdo con cualquiera de las reivindicaciones precedentes, y que incluye el paso, en el caso en que el paso d) marca el fichero como posiblemente viral, de comparar el código de programa con una lista de excepciones permisibles y de suprimir el indicador si se considera que el código de programa está en la lista de excepciones.

6. Un sistema para explorar un fichero informático en busca de infecciones por virus que comprende:

a) medios para identificar el código de programa del fichero;

b) medios para identificar el compilador utilizado para crear el código de programa;

c) medios para determinar la distribución de frecuencias de instrucciones de código máquina seleccionadas o secuencias de dichas instrucciones en el código de programa; y

d) medios para marcar el fichero como posiblemente infectado por un virus, o no, sobre la base de la comparación de la distribución de frecuencias determinada con una distribución de frecuencias de instrucciones de código máquina o de secuencias de las mismas esperada para ese compilador.

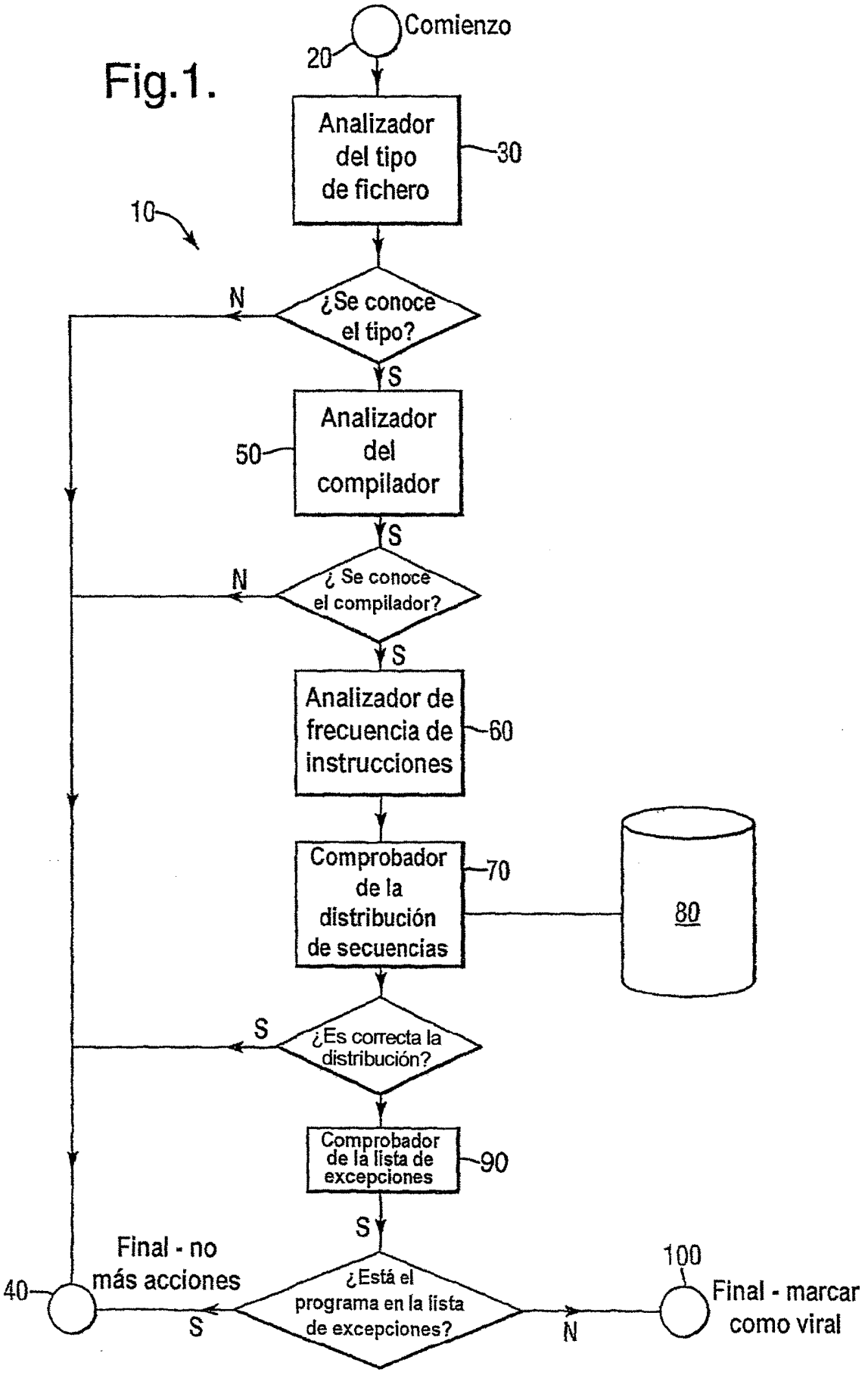
7. Un sistema de acuerdo con la reivindicación 6, en el cual los medios c) para determinar la frecuencia incluyen medios de trazado, pudiendo utilizarse los medios de trazado, trabajando desde un punto de entrada del programa, para trazar un gráfico de ejecución mediante la decodificación de los códigos de operaciones de instrucción sucesivos y la actualización de los conteos de frecuencia de instrucciones decodificadas según avanza este trazado.

8. Un sistema de acuerdo con la reivindicación 7, en el cual los medios de trazado se pueden operar de tal manera que cuando se encuentra una llamada a subrutina o instrucción de bifurcación condicional, el destino de la llamada o de la instrucción de bifurcación es empujado sobre una pila, el trazado avanza al interior de la llamada a la subrutina, y cuando se encuentra una instrucción de retorno, la posición empujada es extraída de la pila y el trazado continúa con las siguientes instrucciones, si existe alguna.

9. Un sistema de acuerdo con la reivindicación 6 u 8, y que incluye medios para examinar el código de programa en busca de construcciones de código de operación, tales como llamada a subrutina o retorno de subrutina, secuencias de instrucciones, las cuales se espera que aparezcan en una proporción entre sí conocida y, si la proporción realmente encontrada se diferencia de la conocida en más de una cierta cantidad, se marca el fichero como posiblemente viral, o se le somete a procesamiento adicional.

10. Un sistema de acuerdo con cualquiera de las reivindicaciones 6 a 9, y que incluye medios, que se puede operar cuando los medios d) marcan el fichero como posiblemente viral, para comparar el código de programa con una lista de excepciones permisibles y suprimir el indicador si se considera que el código fuente está en la lista de excepciones.

Fig.1.


Comienzo
20
Analizador del tipo de fichero
30
10
¿Se conoce el tipo?
N
S
Analizador del compilador
50
S
¿ Se conoce el compilador?
N
S
Analizador de frecuencia de instrucciones
60
Comprobador de la distribución de secuencias
70
80
¿Es correcta la distribución?
S
Comprobador de la lista de excepciones
90
S
Final - no más acciones
40
¿Está el programa en la lista de excepciones?
S
N
100
Final - marcar como viral

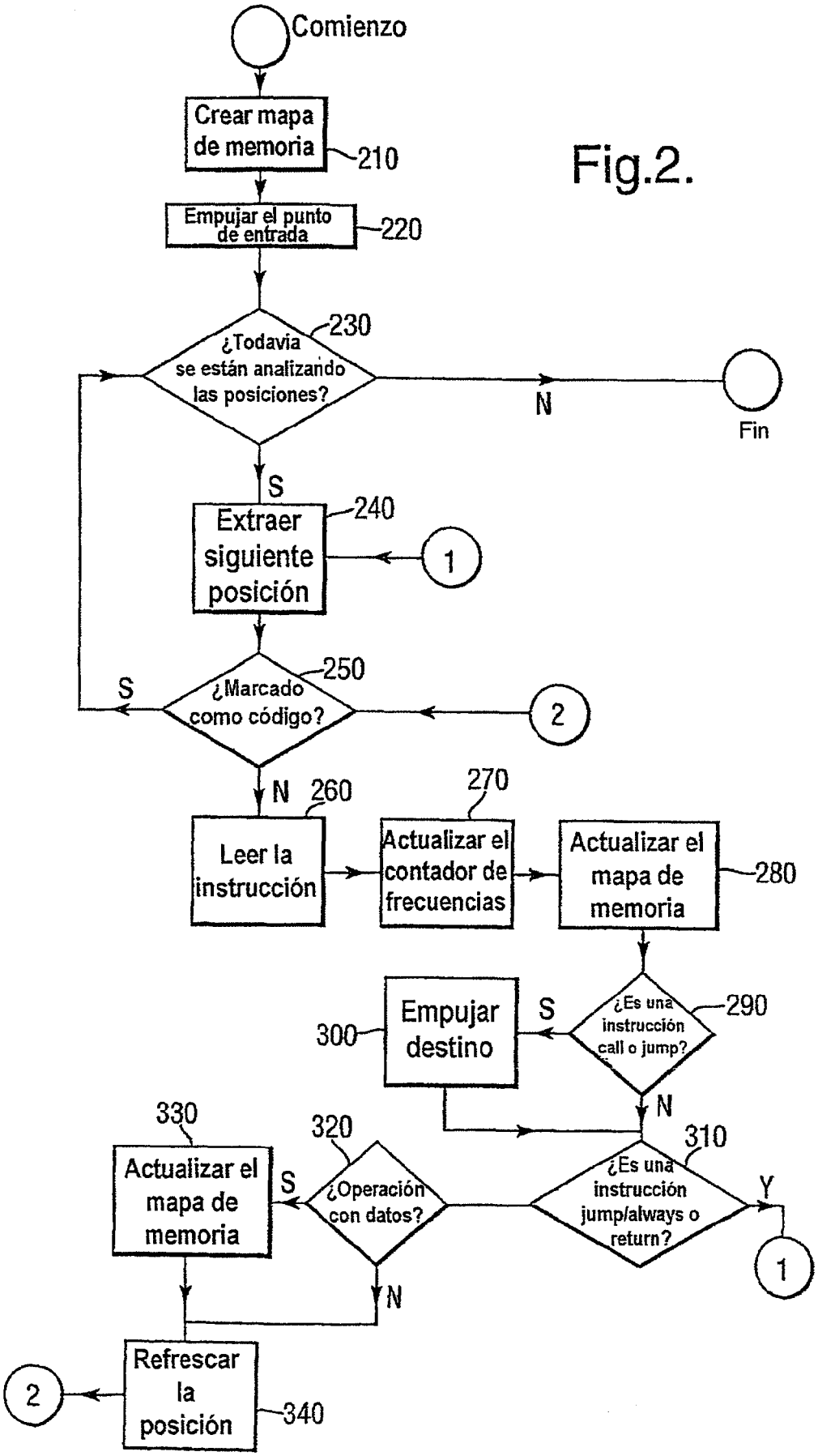

Comienzo
Crear mapa de memoria
210
Empujar el punto de entrada
220
¿Todavia se están analizando las posiciones?
230
N
Fin
S
Extraer siguiente posición
240
1
¿Marcado como código?
250
S
2
N
260
Leer la instrucción
270
Actualizar el contador de frecuencias
Actualizar el mapa de memoria
280
¿Es una instrucción call o jump?
290
S
Empujar destino
300
N
310
¿Es una instrucción jump/always o return?
Y
1
320
¿Operación con datos?
S
330
Actualizar el mapa de memoria
N
Refrescar la posición
340
2

Fig.2.