(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 731 774**

(51) Int. CI.:

G06F 21/54 (2013.01)
G06F 21/57 (2013.01)
H04L 29/06 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **25.09.2014 PCT/US2014/057445**

(87) Fecha y número de publicación internacional: **02.04.2015 WO15048282**

(96) Fecha de presentación y número de la solicitud europea: **25.09.2014 E 14783736 (3)**

(97) Fecha y número de publicación de la concesión europea: **03.04.2019 EP 3049988**

(54) Título: **Sistema y método de solución automática de vulnerabilidades de seguridad**

(30) Prioridad:

25.09.2013 US 201361882347 P

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
19.11.2019

(73) Titular/es:

VERACODE, INC. (100.0%)
65 Network Drive
Burlington, MA 01803, US

(72) Inventor/es:

PAPPAS, THOMAS, MICHAEL

(74) Agente/Representante:

RIZZO , Sergio

DESCRIPCIÓN

Sistema y método de solución automática de vulnerabilidades de seguridad

Referencia cruzada a solicitudes relacionadas

[0001] La presente solicitud reivindica el beneficio y la prioridad de la solicitud de patente provisional estadounidense n.º de serie 61/882,347, titulada "A System and Method for Automated Remedying of Security Vulnerabilities", presentada el 25 de septiembre de 2013.

Campo de la invención

[0002] La presente invención se refiere en general a la detección y mitigación o eliminación de vulnerabilidades en aplicaciones de *software* y, más en concreto, a sistemas y métodos para llevar a cabo estas operaciones directamente en un binario compilado de una aplicación de *software*.

Antecedentes de la invención

[0003] Las aplicaciones informáticas incluyen normalmente fallos o defectos, que hacen que el *software* funcione de forma involuntaria o indeseada. También se pueden explotar algunos defectos para conseguir acceso no autorizado al *software* y/o a los datos asociados al mismo. Normalmente, un experto en la materia puede identificar estos defectos, por ejemplo, probando el sistema de *software* y analizando el código fuente legible para el ser humano del *software*. El experto en la materia puede modificar entonces el código fuente para resolver los defectos, a menudo denominados vulnerabilidades. Este proceso requiere confianza en desarrolladores de aplicaciones expertos y personal de garantía de calidad (QA, por sus siglas en inglés) que tienen un conocimiento especializado de seguridad de *software*. Como tal, el proceso manual de resolución de defectos puede ser costoso. También puede requerir mucho tiempo, y puede extender el plazo de entrega al mercado dentro del ciclo de desarrollo del *software* (SDLC). Además, el proceso manual puede ser propenso a errores. Por ejemplo, una solución en una parte del *software* puede provocar un nuevo defecto inadvertido en otra parte. Por lo tanto, se necesita un sistema y/o un método mejorado para solucionar defectos en aplicaciones de *software*.

[0004] Raymond Mui *et. al.* en "Preventing SQL Injection through Automatic Query Sanitization with ASSIST" - Electronic Proceedings in Theoretical Computer Science, vol. 35, 17 septiembre 2010, pp. 27-38, da a conocer un método para prevenir la inyección de SQL, que es causada cuando un usuario pasa una entrada inadecuada a una base de datos. La técnica proporciona desinfección de consultas automática de consultas SQL mediante el uso de análisis estático y transformación de programa. Baratloo *et. al.* en "Transparent run-time defense against smashing attacks" - Proceedings of the 2000 USENIX Annual Technical Conference - 18-23 junio 2000 - San Diego, CA, EE.UU., pp. 251-262, da a conocer ataques mitigantes contra pilas de procesos con un primer método que intercepta todas las llamadas a las funciones de biblioteca conocidas por ser vulnerables y un segundo método que utiliza modificación binaria de la memoria de proceso para forzar la verificación de los elementos críticos de las pilas antes de su uso.

Sumario de la invención

[0005] Diversos modos de realización de la presente invención pueden detectar y resolver vulnerabilidades en aplicaciones de *software*, al menos en parte, analizando programáticamente uno o más archivos binarios correspondientes a la aplicación de *software* compilada. El análisis de los archivos binarios se puede realizar de forma estática, es decir, sin requerir la ejecución de la aplicación de *software*. Una parte del archivo binario se identifica como asociada a un defecto. En una biblioteca o base de datos de parches se explora un reemplazo, p. ej., un parche binario, para la parte identificada. En algunos casos, la modificación de los archivos binarios de *software* utilizando un parche puede cambiar el comportamiento de la aplicación. En esos casos, o si no se encuentra un parche binario coincidente, se proporciona una notificación a un usuario, que puede abordar manualmente el defecto identificado. Sin embargo, en caso contrario, la parte identificada se reemplaza por el parche, resolviendo, o al menos mitigando, el defecto sin requerir la intervención de personal experto, disminuyendo de esta manera el tiempo de entrega en el SDLC y/o el coste del *software*.

[0006] Por consiguiente, en un aspecto, un método para solucionar automáticamente fallos de seguridad incluye la recepción en memoria de un informe de análisis binario estático para un archivo binario correspondiente a una aplicación. El método también incluye la identificación mediante un procesador de una sección del archivo binario correspondiente a un fallo de seguridad. La identificación puede basarse, al menos en parte, en el informe recibido. El método incluye también la determinación de si existe un parche binario coincidente con la sección identificada en una biblioteca de parches. El parche binario coincidente puede evitar, o al menos mitigar, la vulnerabilidad de seguridad, pero convenientemente el reemplazo no debe causar que el comportamiento del programa cambie de forma considerable. Como tal, el método incluye, si se determina que existe el parche coincidente, la determinación de si el reemplazo de la sección identificada por el parche binario coincidente provocaría un cambio en una función del archivo binario. El método incluye también, si se determina que no se

produce ningún cambio en la función, el reemplazo de la sección identificada por el parche coincidente. En algunos modos de realización, el método incluye opcionalmente la generación de una notificación, p. ej., para alertar a un usuario, si la biblioteca carece de parche binario coincidente o si el reemplazo cambiaría la función.

[0007] La identificación de la sección que corresponde al fallo de seguridad se puede basar además, al menos en parte, en un parámetro especificado, y el parámetro especificado puede incluir uno o más de entre un nombre de función, un tipo de función, un parámetro de función y un tipo de lenguaje. El parámetro de función puede incluir un número de argumentos de función y/o un tipo de al menos un argumento de función. En algunos modos de realización, la identificación de la sección que corresponde al fallo de seguridad incluye la comparación de una parte del archivo binario con una referencia. Por ejemplo, un nombre de una clase, función o método invocado puede compararse con una clase, función o método seguro diferente. La comparación puede ser una comparación exacta o una comparación aproximada, o una combinación de estas.

[0008] En algunos modos de realización, el método incluye la modificación, mediante el procesador, del parche binario coincidente según un contexto del archivo binario, antes del reemplazo de la sección identificada por el parche binario coincidente. Por tanto, la sección identificada se puede reemplazar por un parche binario coincidente modificado. La sección identificada puede incluir una invocación de una función o método o declaración o instanciación de una clase, estructura de datos u objeto. El parche coincidente puede incluir una invocación de otra función o método o declaración o instanciación diferente de otra clase, estructura de datos u objeto diferente, donde la otra función/método o clase/estructura de datos/objeto proporciona una funcionalidad en relación con una funcionalidad proporcionada por la función/método o clase/estructura de datos/objeto que se ha de reemplazar. En algunos modos de realización, las dos funcionalidades son idénticas o son al menos considerablemente similares (p. ej., al menos 99 %, al menos 95 %, al menos 90 %, al menos 60 %). En algunos modos de realización, un número de argumentos de la otra función/método es diferente a un número de argumentos de la función/método que se ha de reemplazar por la otra función.

[0009] En algunos modos de realización, la sección identificada incluye un segmento de código binario que proporciona una funcionalidad de forma no segura, y el parche coincidente incluye otro segmento de código binario que proporciona esa funcionalidad o al menos una funcionalidad considerablemente similar (p. ej., al menos 99 %, al menos 95 %, al menos 90 %, al menos 60 %) de forma segura. En algunos modos de realización, para abordar la vulnerabilidad de seguridad, puede ser necesario desinfectar un dato asociado a la parte del código identificado como no seguro, es decir, comprobar si hay algún componente en el mismo que pueda exponer/explotar una vulnerabilidad de seguridad. Por tanto, en algunos modos de realización, el método incluye la determinación de un contexto de ejecución de la sección identificada y un objeto de datos. El método incluye también la selección de un filtro en función, al menos en parte, del contexto de ejecución determinado. El parche coincidente puede incluir un segmento de código binario: (i) para invocar el filtro seleccionado con el objeto de datos con el fin de obtener un objeto de datos filtrado, y (ii) para proporcionar el objeto de datos filtrado al contexto.

[0010] El método puede incluir añadir otro segmento de código binario que proporciona funcionalidad del filtro seleccionado al archivo binario que corresponde a la aplicación, p. ej., si el uno o más archivos binarios que corresponden a la aplicación no proporcionaron la funcionalidad del filtro seleccionado. En algunos modos de realización, el archivo binario se asocia a Java runtime o a .NET framework, y el contexto incluye uno o más de entre contexto de HTML; contexto de atributos HTML y contexto JavaScript.

[0011] En otro aspecto, un sistema informático incluye un primer procesador y una primera memoria acoplada al primer procesador. La primera memoria incluye instrucciones que, al ser ejecutadas por una unidad de procesamiento que incluye el primer procesador y/o un segundo procesador, programan la unidad de procesamiento para recibir en un módulo de memoria que incluye la primera memoria y/o una segunda memoria acoplada al segundo procesador, un informe de análisis binario estático para un archivo binario que corresponden a una aplicación. Las instrucciones programan también la unidad de procesamiento para identificar una sección del archivo binario, en función de, al menos en parte, el informe recibido. La sección identificada puede corresponderse con un fallo de seguridad. Además, las instrucciones programan la unidad de procesamiento para determinar si existe un parche binario coincidente con la sección identificada en una biblioteca de parches y, si se determina que existe el parche binario coincidente, determinar si se el reemplazo de la sección identificada por el parche binario coincidente provocaría un cambio en una función del archivo binario. Finalmente, las instrucciones programan la unidad de procesamiento para reemplazar la sección identificada con el parche coincidente, si se determina que no se produce ningún cambio en la función. Opcionalmente, en algunos modos de realización, las instrucciones programan la unidad de procesamiento para generar una notificación, si la biblioteca carece de parche binario coincidente o si el reemplazo cambiaría la función. En algunos modos de realización, la segunda memoria acoplada al segundo procesador puede recibir a través de una red la instrucción almacenada en la primera memoria. En diversos modos de realización, las instrucciones pueden programar la unidad de procesamiento para llevar a cabo una o más de las etapas del método descritas anteriormente.

[0012] En otro aspecto, un artículo de fabricación que incluye un soporte de almacenamiento no transitorio tiene instrucciones almacenadas en su interior que, al ser ejecutadas por un procesador, programan el procesador para recibir en memoria acoplada al procesador, un informe de análisis binario estático para un archivo binario que corresponde a una aplicación. Las instrucciones programan también el procesador para identificar una sección del archivo binario, en función de, al menos en parte, el informe recibido. La sección identificada puede corresponderse a un fallo de seguridad. Además, las instrucciones programan el procesador para determinar si existe un parche binario coincidente con la sección identificada en una biblioteca de parches y, si se determina que existe el parche coincidente, determinar si se el reemplazo de la sección identificada por el parche binario coincidente provocaría un cambio en una función del archivo binario. Finalmente, las instrucciones programan el procesador para reemplazar la sección identificada con el parche coincidente, si se determina que no se produce ningún cambio en la función. Opcionalmente, en algunos modos de realización, las instrucciones programan el procesador para generar una notificación, si la biblioteca carece de parche binario coincidente o si el reemplazo cambiaría la función. En diversos modos de realización, las instrucciones almacenadas pueden programar el procesador para llevar a cabo una o más de las etapas del método descritas anteriormente. La invención se define por las reivindicaciones adjuntas.

Breve descripción de los dibujos

[0013] Diversos modos de realización de la presente invención enseñados en el presente documento se ilustran a modo de ejemplo, y no a modo de limitación, en las figuras de los dibujos adjuntos, en las que:

La figura 1 representa esquemáticamente un sistema de parches binario y el entorno operativo de este, según un modo de realización.

La figura 2A representa esquemáticamente un parche que utiliza un filtro para filtrar datos no comprobados, según un modo de realización; y

La figura 2B ilustra la selección de un filtro, según un modo de realización.

Descripción detallada de la invención

[0014] Haciendo referencia a la figura 1, el sistema de parches binario 102 lee en el informe/los resultados de prueba detallados 104 de un análisis binario estático de uno o más archivos binarios 106a-c obtenidos compilando una aplicación de *software*. En algunos modos de realización, los archivos binarios 106a-c corresponden solo a uno o más módulos de la aplicación de *software* y no a toda la aplicación. Ha de entenderse que tres archivos son únicamente ilustrativos, y que tan solo un único archivo y más de tres (p. ej., 5, 10, 40, 100) archivos binarios que puedan representar de forma colectiva la aplicación de software se encuentran dentro del alcance de los diversos modos de realización. El sistema de parches binario 102 puede determinar la localización de los defectos de seguridad de *software* dentro de los binarios compilados analizando los informes, es decir, los resultados de prueba 104. Por ejemplo, los nombres de las funciones/métodos invocados por la aplicación de *software* o un módulo de esta pueden compararse con los nombres en una lista especificada de funciones/métodos que se sabe que son vulnerables a ataques intencionados o accidentales.

[0015] Se pueden determinar entonces una o más localizaciones (p. ej., una localización 110) en el binario donde se invocan una o más de las funciones/métodos que se sabe o se determina que son vulnerables. A menudo, una función que se determina que es vulnerable puede invocarse varias veces en varias localizaciones diferentes. En algunos modos de realización, solo una, unas pocas o todas estas localizaciones se identifican como localizaciones de defectos de seguridad. Una vez se determina una localización de un defecto (p. ej., la localización 110), se identifica una parte (p. ej., una parte 112) del binario que se determina que se asocia al defecto localizado. También se identifican uno o más reemplazos (también denominados parches) 114a-c para esa parte. Ha de entenderse que tres parches son únicamente ilustrativos, y que tan solo un único parche y más de tres (p. ej., 5, 8, 40, 100) parches binarios candidatos y/o parches de reemplazo adecuados se encuentran dentro del alcance de los diversos modos de realización.

[0016] Para identificar parches adecuados, la funcionalidad asociada a la parte identificada puede compararse con alternativas que proporcionen la misma funcionalidad o una funcionalidad similar, pero de forma más segura. Se puede acceder a estas alternativas desde una biblioteca y/o una base de datos de parches 116. La identificación de los defectos, sus localizaciones y/o la selección de los parches de reemplazo puede basarse en uno o más parámetros tales como el lenguaje de código fuente, un nombre/tipo de función/método invocado, un número de parámetros/argumentos de la función/método invocado, tipo o tipos de uno o más argumentos/parámetros de función, etc. Una elección de una alternativa adecuada se puede basar en una comparación exacta (p. ej., reemplazar una generación de un número aleatorio no seguro por una generación de un número aleatorio seguro, reemplazar "strcpy" por "strncpy," etc.).

[0017] En algunos modos de realización, una vez se selecciona un parche adecuado (p. ej., el parche 114a), la parte identificada como asociada al defecto (p. ej., la parte 112) se reemplaza por el parche seleccionado. Por tanto, en lugar de invocar una función/método que pueda proporcionar la funcionalidad requerida de forma no

segura, se puede invocar una función/método diferente que pueda proporcionar considerablemente la misma funcionalidad, pero de forma segura. En algunas situaciones, la invocación de reemplazo requiere datos adicionales. Por ejemplo, si "strcpy" se reemplaza por "strncpy», se debe proporcionar un tamaño de memoria a la nueva llamada de función. Los datos adicionales se pueden obtener a partir de los resultados de prueba, por ejemplo, analizando el contexto de la localización del defecto.

[0018] En algunas situaciones, el código binario que proporciona la funcionalidad de reemplazo puede incluirse ya en el uno o más archivos binarios 106a-106c de la aplicación de *software*. Este código puede incluirse, por ejemplo, en una biblioteca 106b enlazada a uno o más binarios que corresponden a la aplicación de *software*. Como tal, en estas situaciones, simplemente invocar una función/método seguro con cualquier parámetro adicional o diferente según sea necesario, en lugar de invocar la función/método no seguro, puede abordar de forma adecuada la vulnerabilidad. Por lo tanto, en algunos modos de realización, el parche binario de reemplazo modifica solo la invocación de una función/método identificado como vulnerable.

[0019] Sin embargo, en otros casos, el código binario que proporciona la funcionalidad de reemplazo no forma parte del uno o más archivos binarios que corresponden al programa. Por lo tanto, en algunos modos de realización, las localizaciones en las que el código binario que implementa la funcionalidad de las funciones/métodos identificadas como vulnerables se determinan también, y estas localizaciones también se designan como localizaciones de defectos de seguridad. Por ejemplo, el código binario 118 que proporciona la funcionalidad asociada a una llamada de función/método no segura en la localización 110 puede incluirse en el archivo binario 106c, en una localización 120. Por tanto, la localización 120 puede designarse como la localización de un defecto de seguridad además o en lugar de la localización 110.

[0020] En algunos modos de realización, un parche de reemplazo seleccionado 114c incluye no solo un segmento de código binario 122 para modificar la invocación de función/método no segura en la localización 110, sino también otro segmento de código binario 124 que implementa la funcionalidad de la función/método seguro de reemplazo. Se puede reemplazar la invocación 112 en la localización 110 del código defectuoso, como se ha descrito anteriormente, por el segmento de código binario 122. De forma adicional o alternativa, el código defectuoso 118 en sí en la localización 120 puede reemplazarse por la implementación segura 124. En algunos modos de realización, la implementación segura se añade a uno o más archivos binarios 106a-c que se asocian al programa de *software*. En algunos modos de realización un parche binario de reemplazo puede incluir solo una implementación segura de una función/método invocado en un archivo binario.

[0021] Al aplicar un parche binario seleccionado, es decir, al reemplazar en los archivos binarios la parte identificada como defectuosa por un parche, el sistema puede modificar los patrones de instrucciones no seguros para transformarlos en unos más seguros. Por ejemplo, el sistema de parche puede modificar los archivos de clase compilados en un módulo de Java que llama a java/lang/math/Random, que se determina como componente no muy seguro. Las llamadas a java/lang/math/Random se reemplazan por llamadas al módulo java/security/SecureRandom más seguro. De forma similar, un ejecutable C/C++ que llama a "strcpy" de una entrada arbitraria (p. ej., longitud variable) en un *buffer* de tamaño fijo, introduciendo de esta manera una vulnerabilidad, se puede modificar para llamar a "strncpy" en su lugar, con el fin de limitar la longitud de datos copiados en el *buffer* de destino, evitando sobrecargas del *buffer*.

[0022] En algunos modos de realización, los parches se modifican antes de aplicarse a los archivos binarios. Por ejemplo, en el ejemplo strcpy mencionado anteriormente, la aplicación de la corrección strncpy requiere conocimiento sobre la longitud del *buffer* de destino, y el establecimiento de esa longitud como la longitud máxima que se ha de copiar. Parte del resultado del análisis binario incluye algo de contexto sobre los fallos, que en caso de sobrecargas del *buffer* incluye la longitud detectada para el *buffer* de destino. Al utilizar ese valor con la modificación strncpy, se puede asegurar que los datos copiados siempre encajan en el *buffer* de destino. Por tanto, la corrección es como una plantilla a la que se pueden aplicar datos extraídos del análisis (p. ej., la longitud del *buffer*).

[0023] Una selección de una alternativa adecuada también se puede basar en una comparación aproximada. Una coincidencia aproximada de ejemplo puede ser un parche de secuencias de comandos en sitios cruzados (XSS) sensible al contexto que se basa en una coincidencia con el contexto de la localización del fallo. Para ilustrarlo, si la localización del fallo coincide con una prueba de patrón para un contexto de HTML básico, los datos que se generan podrían necesitar solo estar codificados en HTML para corregir el fallo. Si el parcheador no puede hacer coincidir la localización con un patrón en particular, el parcheador puede utilizar por defecto un filtro más estricto, como la codificación de URL (que puede afectar la ejecución/uso del programa).

[0024] Haciendo referencia a la figura 2A, en algunos modos de realización, se determina que en un archivo binario 202, una vulnerabilidad se asocia al uso de datos no comprobados 204, como datos proporcionados por el usuario recibidos en un campo de entrada de una página web. Por tanto, en estos modos de realización, el parche de reemplazo 206 no reemplaza necesariamente una llamada 208 a una función/método por una llamada o una invocación de una función/método de reemplazo. En su lugar, el parche de reemplazo 206 invoca un filtro 210 para comprobar y/o desinfectar datos que se ha de pasar a continuación a otra función/método (p. ej., la

función/método correspondiente a la llamada/invocación 208). El parche de reemplazo 206 puede obtener datos filtrados 212 y puede invocar la función/método mediante una llamada/invocación modificada 214 que utiliza los datos filtrados 212 (p. ej., datos que se determina que no exponen o explotan una vulnerabilidad en la aplicación de *software*), en lugar de los datos no comprobados 204. En diversos modos de realización, el parche adecuado 206 se selecciona en función, al menos en parte, de un entorno de ejecución y/o contexto de la aplicación de *software*.

[0025] Por ejemplo, haciendo referencia a la figura 2B, un sistema de parches binario 252 determina que la aplicación de *software* incluye una vulnerabilidad XSS 254. El sistema 252 también determina si el tiempo de ejecución de la aplicación de *software* se basa en Java 256 o incluye el .NET framework 258. Ha de entenderse que Java runtime y .NET framework son ilustrativos únicamente y que otros tiempos de ejecución, tales como un entorno de ejecución C/C++, Ruby, etc., también se encuentran dentro del alcance de los diversos modos de realización. También se determina mediante el sistema de parches binario 252 un contexto de la localización donde se detectó la vulnerabilidad 254. Algunos ejemplos de contextos incluyen, pero sin carácter limitativo, contextos HTML 260a-b, contextos de atributos HTML 262a-b y contextos JavaScript 264a-b. La naturaleza de la vulnerabilidad que cualquier dato no comprobado puede exponer o explotar en general, aunque no necesariamente, depende del entorno de ejecución y/o del contexto. Como tal, en algunos modos de realización, el sistema 252 selecciona un filtro adecuado para desinfectar los datos no comprobados. Algunos ejemplos de filtros incluyen, pero sin carácter limitativo, codificadores específicos de entorno de ejecución tales como codificadores para HTML 266a-b, codificadores para atributos HTML 268a-b y codificadores JavaScript 270a-b. Como se describe con referencia a la figura 2A, el parche de reemplazo puede invocar las funciones/métodos que pueden ser vulnerables a datos no comprobados utilizando los datos filtrados, p. ej., datos desinfectados, mitigando o eliminando de esta manera estas vulnerabilidades de seguridad.

[0026] En algunas situaciones, no es posible reemplazar por parches todas las partes que se determinan como asociadas a los defectos, p. ej., porque no se encuentra un parche adecuado en la biblioteca. Asimismo, el reemplazo por algunos parches seleccionados puede modificar el comportamiento final de la aplicación. Por ejemplo, se espera que algunas clases de correcciones tengan un impacto en el uso/comportamiento. En el ejemplo de corrección de secuencias de comandos en sitios cruzados una codificación de URL, una página que muestra "Hola, [nombre]" puede modificarse aplicando dos correcciones XSS diferentes. La codificación HTML para el nombre "ABC XYZ" generaría "Hola, ABC XYX", mientras que al utilizar la codificación URL se mostraría en su lugar "Hola, ABC%20XYZ". En estas situaciones, se pueden notificar al usuario los defectos de seguridad y las partes asociadas en los archivos binarios y/o las correspondientes localizaciones del archivo fuente. En algunos modos de realización, se proporciona la notificación del usuario en todos los casos, incluso cuando se identifica un parche adecuado que no se espera que modifique el comportamiento final del *software*, lo que permite al usuario seleccionar qué partes determinadas como defectuosas pueden reemplazarse por los parches identificados.

[0027] En general, el reemplazo de la parte identificada por un parche coincidente puede resolver, o al menos mitigar, el defecto sin requerir la intervención de personal experto, disminuyendo de esta manera el tiempo de entrega en el SDLC y/o el coste de desarrollo del *software*. Un análisis del comportamiento esperado de la aplicación de *software* después del reemplazo puede garantizar, o al menos minimizar, el riesgo de que el reemplazo no sea incompatible con el resto de la aplicación, y que no haya introducido fallos adicionales. La resolución de fallos sin un esfuerzo de desarrollo o con un esfuerzo mínimo puede ser un valor agregado considerable para una seguridad como un sistema de servicio. A estos sistemas también se les pueden incorporar sistemas de evaluación estáticos, dinámicos y/o manuales, que puede utilizarse para probar el binario parcheado con el fin de verificar que la resolución automatizada no modificó sustancialmente el comportamiento originalmente especificado del sistema de *software*. Esto puede ahorrar tanto esfuerzos de desarrollo como esfuerzos de QA. De forma ventajosa, como la ejecución de *software* no es esencial en los sistemas estáticos, la detección y mitigación de vulnerabilidades puede llevarse a cabo a medida que se desarrollan y se compilan algunos componentes de un sistema de *software* grande, antes de que se desarrollen todos los componentes y de que se ensamble todo el *software*.

[0028] Resulta evidente que existen muchas formas de configurar el dispositivo y/o los componentes, interfaces, enlaces de comunicación y métodos del sistema descritos en el presente documento. Los métodos, dispositivos y sistemas dados a conocer se pueden implementar en plataformas de procesador convenientes, entre las que se incluyen servidores de red, ordenadores personales y portátiles y/u otras plataformas de procesamiento. Se pueden contemplar otras plataformas a medida que mejoren las capacidades de procesamiento, incluyendo asistentes personales digitales, relojes informatizados, teléfonos móviles y/u otros dispositivos portátiles. Los métodos y sistemas dados a conocer se pueden integrar en sistemas y métodos de gestión de redes conocidos. Los métodos y los sistemas dados a conocer pueden operar como agente SNMP y pueden configurarse con la dirección IP de una máquina remota que ejecute una plataforma de gestión conforme. Por lo tanto, el alcance de los métodos y sistemas dados a conocer no está limitado por los ejemplos proporcionados en el presente documento, sino que puede incluir el alcance completo de las reivindicaciones y sus equivalentes legales.

[0029] Los métodos, dispositivos y sistemas descritos en el presente documento no están limitados a una configuración de *software* o *hardware* particular, y pueden encontrar aplicabilidad en muchos entornos informáticos o de procesamiento. Los métodos, dispositivos y sistemas se pueden implementar en *hardware* o *software*, o en una combinación de *hardware* y *software*. Los métodos, dispositivos y sistemas se pueden implementar en uno o más programas informáticos, en los que puede entenderse que un programa informático incluye una o más instrucciones ejecutables por procesador. El programa o los programas informático(s) se puede(n) ejecutar en uno o más elementos o máquinas de procesamiento programables, y se pueden almacenar en uno o más medios de almacenamiento legibles por el procesador (entre los que se incluyen elementos de almacenamiento y/o memoria volátil y no volátil), o uno o más dispositivos de entrada, y/o uno o más dispositivos de salida. Por tanto, los elementos/máquinas de procesamiento pueden acceder a uno o más dispositivos de entrada para obtener datos de entrada, y pueden acceder a uno o más dispositivos de salida para comunicar datos de salida. Los dispositivos de entrada y/o salida pueden incluir uno o más de entre los siguientes: memoria de acceso aleatorio (RAM), matriz redundante de discos independientes (RAID), unidad de disquete, CD, DVD, disco magnético, disco duro interno, disco duro externo, memoria extraíble, u otro dispositivo de almacenamiento al que pueda acceder un elemento de procesamiento según se proporciona en el presente documento, donde dichos ejemplos anteriormente mencionados no son exhaustivos, y se proporcionan a modo de ilustración y no de limitación.

[0030] El programa o los programas informático(s) se puede(n) implementar utilizando uno o más lenguajes de programación de alto nivel procedimentales u orientados a objetos para comunicarse con un sistema informático; sin embargo, el programa o los programas se pueden implementar en lenguaje ensamblador o de máquina, si se desea. El lenguaje puede compilarse o interpretarse.

[0031] Como se ha proporcionado en el presente documento, el procesador o los procesadores y/o los elementos de procesamiento pueden estar integrados, por tanto, en uno o más dispositivos que pueden operarse de forma independiente o conjunta en un entorno de red, donde la red puede incluir, por ejemplo, una red de área local (LAN), una red de área extensa (WAN), y/o puede incluir una intranet y/o el Internet y/u otra red. La(s) red(es) puede(n) ser con cable o sin cable o una combinación de estas y puede utilizar uno o más protocolos de comunicaciones para facilitar las comunicaciones entre los diferentes procesadores/elementos de procesamiento. Los procesadores pueden configurarse para un procesamiento distribuido y pueden utilizar, en algunos modos de realización, un modelo de cliente-servidor, según sea necesario. Por consiguiente, los métodos, dispositivos y sistemas pueden utilizar múltiples procesadores y/o dispositivos de procesador, y las instrucciones del procesador/elemento de procesamiento pueden dividirse entre dicho único o múltiples procesadores/dispositivos/elementos de procesamiento.

[0032] El dispositivo o los dispositivos o los sistemas informáticos que se integran en el/los procesador(es)/elemento(s) de procesamiento pueden incluir, por ejemplo, un ordenador personal o varios, una estación de trabajo (p. ej., Dell, HP), un asistente personal digital (PDA), un dispositivo portátil tal como un teléfono móvil, un ordenador portátil, un dispositivo de mano u otro dispositivo que pueda integrarse en uno o varios procesadores que puedan operar como se proporciona en el presente documento. Por consiguiente, los dispositivos proporcionados en el presente documento no son exhaustivos y se proporcionan para fines de ilustración y no de limitación.

[0033] Se puede entender que las referencias a «un procesador», o «un elemento de procesamiento», «el procesador» y «el elemento de procesamiento» incluyen uno o más microprocesadores que pueden comunicarse en uno o varios entornos independientes y/o distribuidos, y pueden por tanto configurarse para comunicarse mediante comunicaciones con cable o inalámbricas con otros procesadores, donde dichos uno o más procesadores pueden configurarse para operar en uno o más dispositivos controlados por procesador/elementos de procesamiento que pueden ser dispositivos similares o diferentes. Por tanto, también puede entenderse que el uso de la terminología de «microprocesador», «procesador» o «elemento de procesamiento» incluye una unidad de procesamiento central, una unidad lógica aritmética, un circuito integrado (IC) de aplicación específica, y/o un motor de tareas, estando estos ejemplos proporcionados a modo de ilustración y no de limitación.

[0034] Además, las referencias a la memoria, a menos que se especifique lo contrario, pueden incluir uno o más elementos y/o componentes de memoria accesibles y legibles por procesador que pueden ser internos al dispositivo controlado por procesador, externos al dispositivo controlado por procesador, y/o se puede acceder a ellos mediante una red con cable o inalámbrica utilizando varios protocolos de comunicaciones y, a menos que se especifique lo contrario, pueden disponerse para que incluyan una combinación de dispositivos de memoria externos e internos, donde dicha memoria puede ser contigua y/o con particiones en función de la aplicación. Por ejemplo, la memoria puede ser una unidad flash, un disco, CD/DVD, memoria distribuida, etc. Las referencias a las estructuras incluyen enlaces, colas, gráficos, árboles, y estas estructuras se proporcionan a modo de ilustración y no de limitación. Se puede entender que las referencias en el presente documento a instrucciones o instrucciones ejecutables, de conformidad con lo anterior, incluyen *hardware* programable.

[0035] Aunque los métodos y sistemas se han descrito en relación con modos de realización específicos de los mismos, no están limitados a estos. Como tal, pueden resultar evidentes muchas modificaciones y variaciones

teniendo en cuenta la información dada a conocer anteriormente. Los expertos en la materia pueden realizar muchos cambios adicionales en los detalles, los materiales y la disposición de partes descritos e ilustrados en el presente documento. Por consiguiente, se entenderá que los métodos, los dispositivos y los sistemas proporcionados en el presente documento no han de limitarse a los modos de realización dados a conocer en el presente documento, pueden incluir prácticas distintas a las descritas específicamente, y han de interpretarse de la forma más amplia que permita la ley.

REIVINDICACIONES

1. Método para solucionar automáticamente fallos de seguridad, comprendiendo el método las etapas consistentes en:

recibir en memoria un informe de análisis binario estático (104) para un archivo binario (106a-c; 202) que corresponde a una aplicación, identificando el informe un fallo de seguridad (110) en una sección (112) del archivo binario;

identificar, mediante un procesador, en función de, al menos en parte, el informe recibido, la sección (112) del archivo binario que corresponde con el fallo de seguridad (110), comprendiendo la sección identificada un primer segmento de código binario (118) que proporciona una funcionalidad de forma no segura;

localizar un parche binario coincidente (114a-c; 206) que comprende un segundo segmento de código binario que proporciona funcionalidad de forma segura y que coincide con la funcionalidad proporcionada por el primer segmento de código binario (118) en la sección identificada, en una biblioteca de parches (116);

determinar si el reemplazo dentro del archivo binario (106a-c) de la sección identificada (112) por el parche binario coincidente (114a-c; 206) daría lugar a un cambio en una función del archivo binario;

si se determina que no se produce ningún cambio en la función, reemplazar la sección identificada (112) del archivo binario (106a-c; 202) por el parche coincidente (114a-c); y

en caso contrario, generar una notificación de que el reemplazo cambiaría la función.

2. Método de la reivindicación 1, donde la identificación de la sección (112) que corresponde al fallo de seguridad (110) se basa además, al menos en parte, en un parámetro especificado.

3. Método de la reivindicación 2, donde el parámetro especificado comprende al menos uno de entre un nombre de función, un tipo de función, un parámetro de función y un tipo de lenguaje.

4. Método de la reivindicación 3, donde el parámetro de función comprende al menos uno de entre un número de argumentos de función y un tipo de al menos un argumento de función.

5. Método de la reivindicación 1, donde la identificación de la sección (112) que corresponde al fallo de seguridad (110) comprende la comparación de una parte del archivo binario (106a-c; 202) con una referencia.

6. Método de la reivindicación 5, donde la comparación es una de entre una comparación exacta y una comparación aproximada.

7. Método de la reivindicación 1, que comprende además la modificación, mediante el procesador, del parche binario coincidente (114a-c; 206) según un contexto del archivo binario (106a-c; 202), antes del reemplazo de la sección identificada (112).

8. Método de la reivindicación 1, donde:

la sección identificada (112) comprende una invocación de una función; y

el parche coincidente (114a-c) comprende una invocación de otra función diferente, proporcionando una funcionalidad en relación con una funcionalidad proporcionada por la función.

9. Método de la reivindicación 8, donde un número de argumentos de la otra función es diferente a un número de argumentos de la función.

10. Método de la reivindicación 1, que comprende además:

la determinación de un contexto de ejecución de la sección identificada (112) y un objeto de datos; y

la selección de un filtro (210) en función, al menos en parte, del contexto de ejecución determinado;

donde el parche coincidente (206) comprende un segmento de código binario: (i) para invocar el filtro seleccionado (210) con el objeto de datos con el fin de obtener un objeto de datos filtrado (212), y (ii) para proporcionar el objeto de datos filtrado (212) al contexto.

11. Método de la reivindicación 10, que comprende además añadir otro segmento de código binario que proporciona funcionalidad del filtro seleccionado al archivo binario que corresponde a la aplicación.

12. Método de la reivindicación 10, donde el archivo binario (202) se asocia a Java runtime o a .NET framework, y el contexto comprende al menos uno de entre:

un contexto de lenguaje de marcado de hipertexto, HTML; un contexto de atributos HTML; y un contexto JavaScript.

13. Sistema que comprende:

un primer procesador; y

una primera memoria acoplada al primer procesador, comprendiendo la primera memoria instrucciones que, al ser ejecutadas por una unidad de procesamiento que comprende al menos uno de entre el primer procesador y un segundo procesador, programan la unidad de procesamiento, para solucionar automáticamente fallos de seguridad según el método de cualquiera de las reivindicaciones anteriores.

14. Artículo de fabricación que comprende un soporte de almacenamiento no transitorio que tiene instrucciones almacenadas en su interior que, al ser ejecutadas por un procesador, programan el procesador para solucionar automáticamente fallos de seguridad según el método de cualquiera de las reivindicaciones 1 a 12.

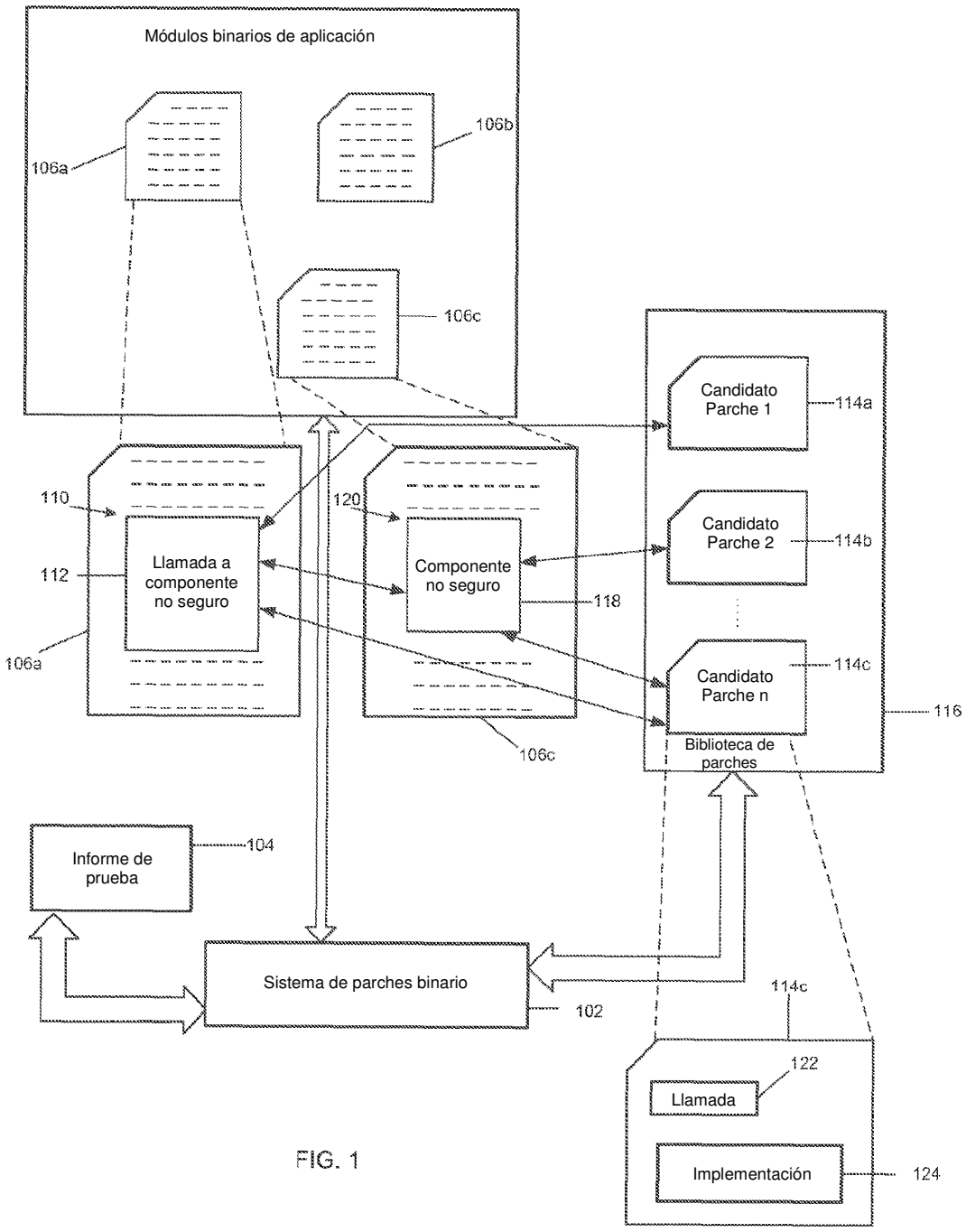

Módulos binarios de aplicación
106a
106b
106c
Candidato Parche 1
114a
Candidato Parche 2
114b
Candidato Parche n
114c
116
Biblioteca de parches
110
Llamada a componente no seguro
112
106a
120
Componente no seguro
118
106c
Informe de prueba
104
Sistema de parches binario
102
114c
122
Llamada
Implementación
124

FIG. 1

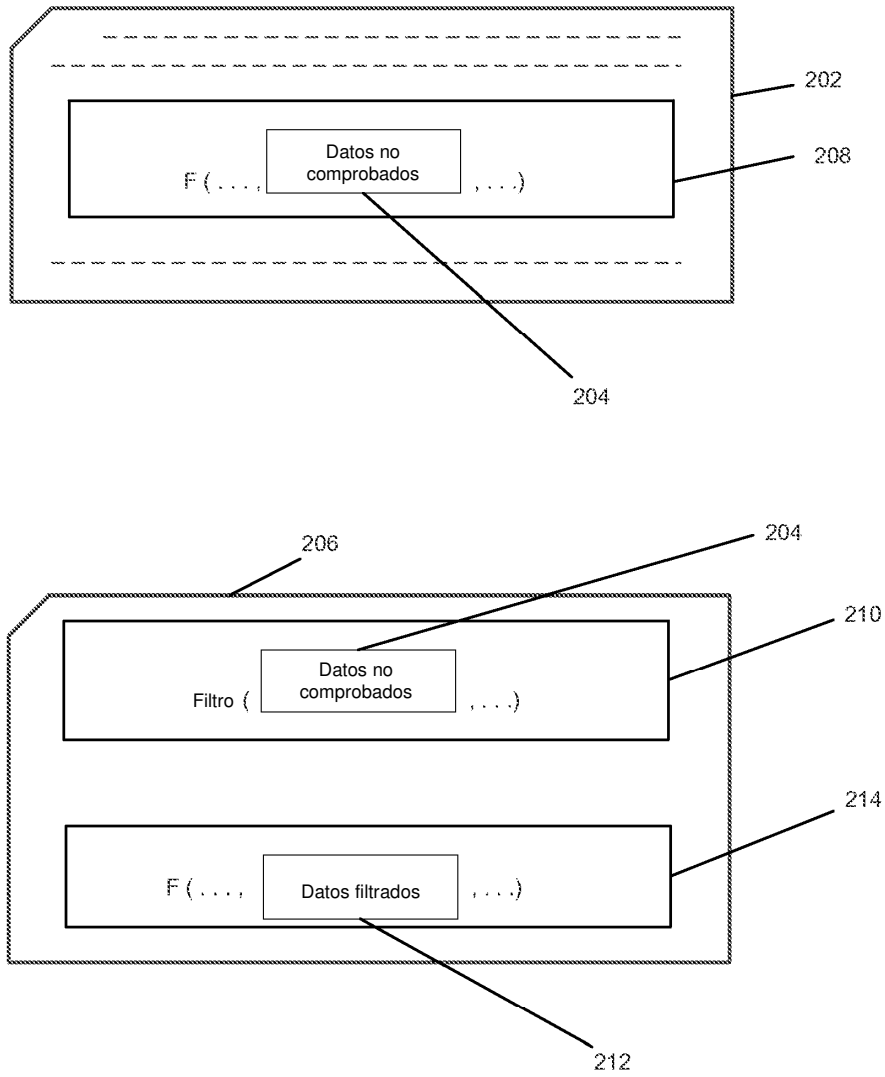
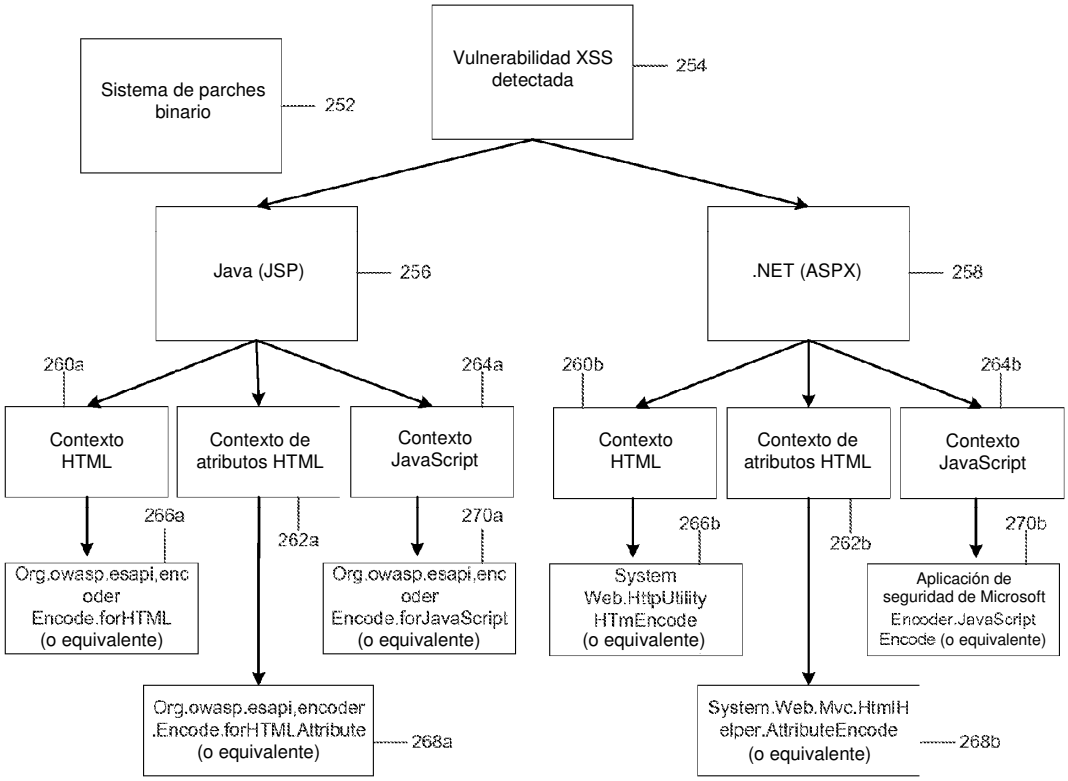

202
208
F (. . . ,
Datos no comprobados
, . . .)
204
206
204
210
Filtro (
Datos no comprobados
, . . .)
214
F (. . . ,
Datos filtrados
, . . .)
212

FIG. 2A

FIG 2B


Vulnerabilidad XSS detectada
254
Sistema de parches binario
252
Java (JSP)
256
.NET (ASPX)
258
260a
264a
260b
264b
Contexto HTML
Contexto de atributos HTML
Contexto JavaScript
Contexto HTML
Contexto de atributos HTML
Contexto JavaScript
266a
262a
270a
266b
262b
270b
Org.owasp.esapi,encoder Encode.forHTML (o equivalente)
Org.owasp.esapi,encoder Encode.forJavaScript (o equivalente)
System Web.HttpUtility HTmEncode (o equivalente)
Aplicación de seguridad de Microsoft Encoder.JavaScript Encode (o equivalente)
Org.owasp.esapi,encoder .Encode.forHTMLAttribute (o equivalente)
268a
System.Web.Mvc.HtmlHelper.AttributeEncode (o equivalente)
268b