REVISTA DIGITAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES
TELEM@TICA
Revista Telem@tica. Vol. 13. No. 1, enero-abril, 2014, p. 63-71 ISSN 1729-3804

HERRAMIENTA PARA LA DETECCIÓN DE POSIBLES VULNERABILIDADES EN EL CÓDIGO FUENTE DEL REPOSITORIO DE GNU/LINUX NOVA

TOOL FOR DETECTING VULNERABILITIES IN SOURCE CODE REPOSITORY GNU / LINUX NOVA

Abel Alfonso Fírvida Donéstevez1, Adrian Serafin Lamadrid Cuesta2, Edilberto Blez Deroncelé3, Allan Pierra Fuentes4
1 Universidad de las Ciencias Informáticas, Cuba, aafirvida@uci.cu
2 Segurmatica, Cuba, lamadrid@segurmatica.cu
3 Universidad de las Ciencias Informáticas, Cuba, eblez@uci.cu
3 Universidad de las Ciencias Informáticas, Cuba, apierra@uci.cu

RESUMEN: Con el desarrollo de la presente investigación, se logró elevar la seguridad de cada uno de los procesos que intervienen en la construcción, actualización y configuración de los repositorios del sistema operativo GNU/LINUX Nova, a partir de la detección de posibles vulnerabilidades presentes en el código fuente de las aplicaciones que interactúan en dichos repositorios, permitiendo esto que los usuarios finales puedan descargar di-chas aplicaciones sin comprometer la seguridad de sus estaciones de trabajo.
En el escenario actual del desarrollo de la distribución de GNU/Linux Nova, se incluyen los procesos de construcción y actualización del repositorio de paquetes. Este se construye partiendo del código fuente de las aplicaciones, las cuales son compiladas para obtener los binarios que finalmente conforman dicho repositorio. Un punto neurálgico en el proceso de desarrollo de Nova, es el poder detectar de manera automática, posibles vulnerabilidades existentes en dicho código fuente. Con vistas a lograr dicho objetivo se desarrolló una herramienta capaz de identificar las posibles vulnerabilidades de tipo desbordamiento de buffer, desbordamiento de pila y ataques de código remoto en el código fuente de las aplicaciones; desarrollando patrones de búsquedas léxicas y de conexiones de red abiertas.
Palabras Clave: software libre, nova, repositorio, detección de vulnerabilidades, código fuente.

ABSTRACT: With the development of this research, we managed to increase the security of each of the processes involved in the construction, upgrading and configuring repositories operating system GNU / LINUX Nova, from identifying potential vulnerabilities in the source applications that interact in such repositories, allowing it to end users can download these applications without compromising the security of their workstations. In the current stage of development of the GNU / Linux Nova include construction processes and update package repository. This is constructed starting from the application source code, which are compiled binaries for finally make that repository. A key point in the development process Nova, is able to automatically detect, potential vulnerabilities in the source code. In order to achieve this objective we developed a tool to identify potential vulnerabilities type buffer overflow, stack overflow and remote code attacks in the application source code, develo-ping lexical search patterns and open network connections.
KeyWords: Free software, nova, repository, exploit detection, source code.

1. INTRODUCCIÓN

La Seguridad Informática es una disciplina cuya importancia crece día a día. Aunque la seguridad es un concepto difícil de medir, su influencia afecta directamente a todas las actividades de cualquier entorno informatizado en los que interviene el ingeniero en informática, por lo que es considerada de vital importancia.

Se define la seguridad como "una característica de cualquier sistema (informático o no) que nos indica que ese sistema está libre de todo peligro, daño o riesgo, y que es, en cierta manera, infalible"... "para el caso de sistemas informáticos, es muy difícil de conseguir (según la mayoría de los expertos, imposible) por lo que se pasa a hablar de confiabilidad". [1]

La misma se conforma de tres elementos puntuales:

1. Integridad: garantiza que se proteja la exactitud y totalidad de los datos y los métodos de procesamiento.

2. Confidencialidad: garantiza que la información sea accesible sólo a las personas autorizadas.

3. Disponibilidad: garantiza que los usuarios autorizados tengan acceso a la información y a los recursos cuando los necesiten.

Actualmente se puede entender que un sistema informático seguro, tal como un Sistema Operativo (SO), no es solo aquel que implementa políticas de seguridad durante la fase de desarrollo, sino que contempla durante todo el proceso de concepción del sistema informático políticas que permitirán minimizar posibles vulnerabilidades en el futuro.

En la facultad 1 de la UCI tiene lugar el Centro de Software Libre (CESOL), donde se ha venido llevando a cabo el desarrollo del SO "Nova"; una distribución de GNU/Linux desarrollada por estudiantes y profesores, con la participación de miembros de otras instituciones.

Durante el desarrollo y mantenimiento del SO Nova tienen lugar importantes procesos y fases las cuales deben ser aseguradas por el impacto que estas tienen sobre el producto final. Uno de estos procesos es el de construcción de los repositorios, el cual tiene gran importancia ya que a través de los repositorios se le brinda la posibilidad al usuario de interactuar con los programas que desee.

En el proceso de construcción de los repositorios tiene lugar otro importante subproceso "El empaquetamiento de las fuentes". Durante este subproceso el código fuente de las aplicaciones que se desean empaquetar es transformado en un instalador, todo ello a través de mecanismos que automatizan los procedimientos. Luego de estar empaquetadas, sus respectivos paquetes o instaladores son enviados a los repositorios correspondientes, de acuerdo a la categoría de la herramienta desarrollada.

Al apreciar la manera en que se llevan a cabo todos los procesos relacionados con la construcción de los repositorios, se puede entender que los repositorios son grandes contenedores muy eficientes de aplicaciones desarrolladas y empaquetadas en un formato específico llamado .deb. Sin embargo el proceso de construcción de repositorios puede ser inseguro siempre que las aplicaciones contenidas en

él lo sean. Al hablar de sistema "inseguro " se hace referencia a un sistema no confiable, no auditado, no controlado o mal administrado con respecto a la seguridad.

La auditoría de código tiene su origen en la necesidad de detectar malas prácticas de programación en las cuales frecuentemente los programadores de aplicaciones incurren, en ocasiones por desconocimiento y en otras por comodidad y aparente simpleza. La búsqueda de vulnerabilidades a partir del código fuente corresponde a una de las técnicas que se pueden aplicar para lograr una programación segura, la cual a su vez está enfocada en lograr una alta seguridad informática en las aplicaciones. Por tanto, esta investigación parte fundamentada por la idea de poder desarrollar una herramienta que permita auditar tanto código fuente como compactados de los programas que se desarrollen en los lenguajes C, C++, Pearl y Python, permitiendo esto que el usuario pueda conocer el nivel de seguridad con que cuenta el software que está instalando desde un repositorio, que los desarrolladores de software puedan chequear cuan confiable es su código y que las instituciones, que gestionen repositorios de distribuciones de GNU/Linux, puedan comprobar el nivel de seguridad de las aplicaciones que vayan a publicar o ya se encuentren disponibles en sus repositorios.

2. CONTENIDO

2.1 Metodología

La metodología empleada para el desarrollo de la aplicación es la metodología Nova OpenUp, la cual constituye una metodología de desarrollo de distribuciones GNU/Linux. La misma abarca todo el ciclo de vida de proyectos que hacen productos de este tipo mediante la ejecución de un conjunto de disciplinas compuestas por actividades que se relacionan entre sí, roles y artefactos. Esta es orientada a la ejecución del proceso definiendo un conjunto de acciones que brindan gran importancia a las personas desde el punto de vista de lo que estas puedan aportar al desarrollo de las distribuciones GNU/Linux manteniendo la comunicación bidireccional. También está orientada a las tareas necesarias para obtener un producto de calidad. Esta tiene en cuenta las buenas prácticas que sugieren metodologías de nombre como OpenUp, XP, y otras que corresponden al nivel 2 de CMMI. Ésta metodología tiene como principios:

1. Colaborar para sincronizar intereses y compartir conocimiento: Este principio promueve prácticas que impulsan un ambiente de equipo saludable, que facilite la colaboración, la socialización y exteriorización de conocimientos.

2. Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto: Este principio promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los beneficios obtenidos por los participantes y que cumple con los requisitos y restricciones del proyecto.

3. Centrarse en la Arquitectura de forma temprana: Permite minimizar el riesgo y organizar el desarrollo, determinando llevar a cabo una arquitectura incremental.

4. Desarrollar evolutivamente para obtener retroalimentación y mejoramiento continuo: Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continúa de los participantes del proyecto, permitiendo demostrarles incrementos.

5. Utilizar una infraestructura tecnológica adecuada que fomente el trabajo en entornos de software libre: Siendo la premisa fundamental maximizar la colaboración entre todos los involucrados.

6. Aplicar métodos para la obtención de calidad en procesos y productos: Teniendo en cuenta las políticas que define el programa de mejora emitido por CaliSoft (departamento de prueba y control de la calidad en la UCI) basado en las prácticas que propone CMMI para obtener el nivel 2.

2.2 Especificaciones

Para obtener la herramienta deseada, se implementaron 4 métodos principales:

1. El método AuditPython, para buscar vulnerabilidades en los ficheros escritos en lenguaje de programación Python.

2. El método AuditC, para buscar vulnerabilidades en los ficheros escritos en los lenguajes de programación C y C++.

3. El método AuditPerl, para buscar vulnerabilidades en el los ficheros escritos en el lenguaje de programación Perl.

4. El método Descompresor, para descomprimir los ficheros comprimidos y colocarlos en el directorio de salida y allí pasar a la búsqueda de vulnerabilidades.

2.2.1 Método AuditPython

Esta funcionalidad se encarga de recorrer el fichero que recibe como parámetro y que contiene código fuente escrito en lenguaje de programación python, buscando de esta forma funciones que se usen mal, que se encuentren contraindicadas o que puedan generar desbordamientos de buffers. De igual manera localiza cualquier intento de acceso a recursos críticos del sistema, conexiones que se abran vía sockets, llamadas al sistema, entre otras vulnerabilidades, cuyas detecciones fueron definidas como requerimientos.

Este método genera un fichero con el mismo nombre, pero con la extensión .bug, ubicado en el mismo directorio del fichero que se encuentra en revisión en caso de existir vulnerabilidades y escribe allí toda la información respecto a las mismas. Además de almacenar en un fichero temporal la dirección del fuente vulnerable, para mostrarle al usuario al concluir todas las auditorías, de esta modo el usuario puede conocer cuales de todos los ficheros revisados fueron vulnerables.

2.2.2 Método AuditC

Esta funcionalidad se encarga de recorrer el fichero que recibe como parámetro y que contiene código fuente escrito en lenguaje de programación C o C++, buscando de esta forma funciones que se usen mal, que se encuentren contraindicadas o que puedan generar desbordamientos de buffers. De igual manera localiza cualquier intento de acceso a recursos críticos del sistema, conexiones que se abran vía sockets, llamadas al sistema, mal uso de funciones de impresión, y de copia a nivel de bytes, entre otras vulnerabilidades, cuyas detecciones fueron definidas como requerimientos. Este método genera un fichero con el mismo nombre, pero con la extensión bug, ubicado en el mismo directorio del fichero que se encuentra en revisión en caso de existir vulnerabilidades y escribe allí toda la información respecto a las mismas. De igual forma almacena en un fichero temporal la dirección del fuente vulnerable, para

mostrarle al usuario al concluir todas las auditorías, de esta modo el usuario puede conocer cuáles de todos los ficheros revisados fueron vulnerables.

2.2.3 Método AuditPerl

Esta funcionalidad se encarga de recorrer el fichero que recibe como parámetro y que contiene código fuente escrito en lenguaje de programación Perl, buscando de esta forma funciones que se usen mal, que se encuentren contraindicadas o que puedan generar desbordamientos de buffers. De igual manera localiza cualquier intento de acceso a recursos críticos del sistema, conexiones que se abran vía sockets, llamadas al sistema, redirecciones de páginas web, condiciones de carreras en algunas funciones, entre otras vulnerabilidades, cuyas detecciones fueron definidas como requerimientos. Este método genera un fichero con el mismo nombre, pero con la extensión .bug, ubicado en el mismo directorio del fichero que se encuentra en revisión en caso de existir vulnerabilidades y escribe allí toda la información respecto a las mismas, además de almacenar en un fichero temporal la dirección del fuente vulnerable, para mostrarle al usuario al concluir todas las auditorías, de esta modo el usuario puede conocer cuáles de todos los ficheros revisados fueron vulnerables.

2.2.4 Método Descompresor

La función del método Descompresor es la de extraer todos aquellos ficheros que estén comprimidos en cualquiera de los siguientes formatos (rar, ar, zip,tar.gz, jar,tar.bz2). Para ello ejecuta un plugin encargado de descomprimir al fichero en dependencia del formato de este. El que estará almacenado en una carpeta titulada "plugins" ubicada en el mismo directorio de la herramienta. Cada plugin se llamará con el nombre del formato que descomprime antecedido de la letra d, por ejemplo: el plugin que descomprime un rar, se llamará drar, de esta manera el método Descompresor sabrá como ejecutar cada uno. Estos plugins implementaran la forma en que cada uno de estos ficheros deben ser descomprimidos. Luego de haberlo hecho, el método ejecutará las auditorías que correspondan a cada uno de los ficheros ubicados dentro de los nuevos directorios extraídos, que habrán sido creados en una ubicación especificada como directorio de salidas.

2.3 Resultados y Discusión

Actualmente existen diversas herramientas que se pueden utilizar para auditar código fuente. A fin de poder conocer las características y funcionalidades principales con las que cuentan dichas herramientas, se desarrolló un estudio sobre las mismas; enfocado fundamentalmente sobre las de código abierto que se encuentran disponibles en el repositorio de Nova. Flawfinder es una herramienta escrita en lenguaje Python por David Wheeler, el autor de la "Programación Segura para Linux". Su primera versión fue la 0.12 y la más actual es la 1.26, diseñadas para las plataformas : GNU/Linux, UNIX, Windows, OS/2, Mac. Flawfinder está orientada a analizar código fuente escrito en los lenguajes C y C++, teniendo como principal objetivo la detección de potenciales debilidades de seguridad que se llevan a cabo durante la escritura del código fuente. El programa funciona escaneando el código y buscando el uso de las funciones que tiene en su base de datos de funciones que normalmente se usan mal. Cuando se hace funcionar sobre un directorio que contiene código fuente, se obtiene un informe de los problemas potenciales que ha detectado, ordenados por riesgo (riesgo es un entero de 1 a 5). Para obviar los riesgos menores, es posible decirle al programa que no informe sobre debilidades menores de un nivel de riesgo particular. De forma predefinida, la salida aparecerá en texto plano, pero también hay disponible un informe en HTML.[2]. Para facilitar la lectura del informe, se puede indicar que contenga

la línea en la que se usa la función, lo que puede ser útil para detectar un problema inmediatamente, ya que de otra forma podría ser imposible. Examina el código fuente buscando fallos de seguridad (de ahí el nombre). Busca funciones con problemas conocidos. Como desbordamientos de buffer (strcpy(), scanf(), y otras por el estilo), errores de formato (printf (), syslog ()) condiciones de carrera como son (access(), chmod(), entre otros), posibles problemas de uso de metacaracteres en la shell (exec(), system()) y generadores de números aleatorios (random ()), la aplicación nos dará un informe con todos los posibles errores.[3]

Herramienta: RATS: RATS(Rouge Auditing Tool for Security)

RATS o ratas como también se le conoce es una herramienta de auditoría áspera para la funcionalidad de Ratas de Seguridad. Fue escrita en lenguaje C por la empresa Software Seguro Inc, disponible para las plataformas: GNU/Linux, UNIX y Windows Commandline y es muy útil como primer paso para realizar una revisión de un código[4]. Sus versiones (desde la primera 0.9, hasta la más actual 2.1), se encuentran bajo la licencia GNU GPL v2. RATS es muy cercana a Flawfinder, con la excepción de que admite un mayor abanico de lenguajes. En la actualidad, tiene soporte para C, C++, Perl, PHP y Python. Ambas herramientas estuvieron desarrolladas simultáneamente. Cuándo los desarrolladores respectivos descubrieron la otra herramienta, estos decidieron liberar la primera versión de cada herramienta exactamente el mismo día. RATS es una de las pocas herramientas que soporte los lenguajes de programación Perl, PHP y Python. Para cada vulnerabilidad encontrada imprime una explicación del problema (si esta disponible en la base de datos de vulnerabilidades). Finalmente RATS imprime el número de las líneas analizadas y el tiempo que utilizó. La herramienta usa un archivo XML sencillo para leer sus vulnerabilidades, lo que la convierte en una de las herramientas más sencillas para hacer modificaciones. Fácilmente se pueden añadir funciones nuevas para cada uno de los lenguajes soportados.[5]

Herramienta: Pscan

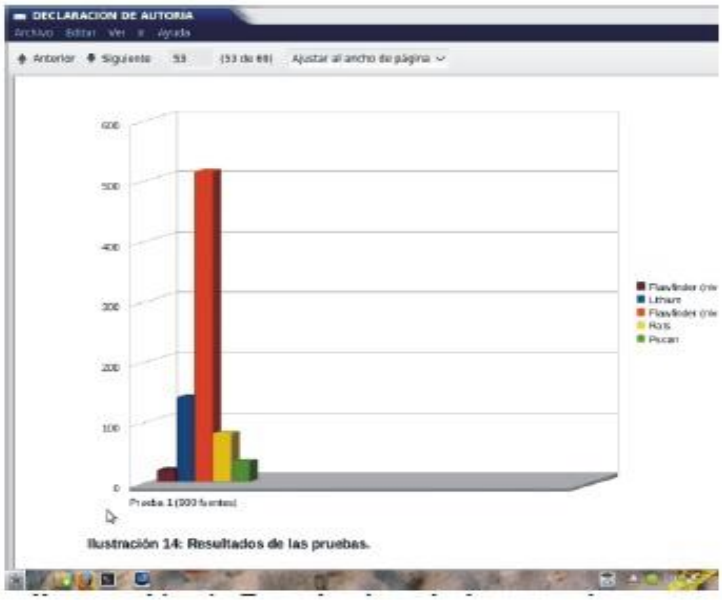
Pscan difiere de las herramientas anteriores en que no es un analizador de propósito general. Se trata de un programa específico para ayudar a detectar errores de cadena de formato. La herramienta intentará encontrar incidencias en el uso de funciones en código fuente C y C++, como printf, fprintf y syslog. Los errores de cadena de formato son bastante sencillos de detectar y corregir. A pesar de que sean el tipo más reciente de ataques de software, la mayoría de ellos ya se pueden descubrir y reparar.[6]

ITS4 es una herramienta que pertenece a la sección non-free (no libre) del archivo de Debian, y sólo está disponible para la antigua distribución estable. ITS4 se puede usar para buscar potenciales agujeros de seguridad en C y C++, al igual que flawfinder. La salida que produce pretende ser inteligente, ya que no tiene en cuenta algunos de los casos en los que las llamadas a las funciones peligrosas se han hecho con cuidado.

A fin de poder conocer la validez del componente desarrollado, se aplicó un proceso de caja negra sobre los repositorios de fuentes. Los fuentes se encontraban comprimidos en formato tar.gz, de modo que la herramienta antes de realizar las rutinas de búsquedas de vulnerabilidades, realizó las rutinas correspondientes de descompresión.

Al realizar dicha prueba con las herramientas rats, flawfinder, lithium y pscan se obtuvo que el tiempo de ejecución para el caso de flawfinder (nivel 4) fue de 3 minutos llegando a detectar 514 vulnerabilidades entras las cuales muchas eran de baja importancia, mientras que al ser ejecutada la misma herramienta en su nivel 5 solo fueron detectadas 19 vulnerabilidades, en este caso de alto impacto.(rast)

Para el caso de lithium el tiempo de ejecución fue de 1 minuto llegando a detectar 131 vulnerabilidades. La herramienta rats culminó en 1 minuto con 80 vulnerabilidades encontradas y para pscan también se tardó 1 minuto, detectando 34 vulnerabilidades. Por tanto al analizar los resultados de la prueba, se puede apreciar que la herramienta lithium realiza el chequeo de vulnerabilidades en un tiempo aceptable, y detecta un número considerablemente elevado de vulnerabilidades de alto y medio impacto en la seguridad de la aplicación, como se puede observar en la fígura 1.



Fígura 1: Resultados de las pruebas

CONCLUSIONES

Al culminar la presente investigación, se logró desarrollar una herramienta capaz de detectar un mayor número de posibles vulnerabilidades en los paquetes de código fuente almacenados en los repositorios del sistema operativo GNU/Linux Nova, logrando así detectar debilidades en las aplicaciones que hasta el momento no eran detectadas y por consiguiente reducir el número de las mismas, haciendo a los repositorios de Nova más confiables.

REFERENCIAS BIBLIOGRÁFICAS

1. "IRAM-ISO-IEC 17799 Tecnología de la información. Código de práctica para la gestión de la seguridad de la información". [en línea] [Fecha de consulta: 2012].

2. DWHEELEER: "FlawFinder" [en línea]. Actualizada: 11 febrero 2012. [Fecha de consulta: 5 marzo 2012]:Disponible en: http://www.dwheeler.com/flawfinder/.

3. BUGBLOG: "Blog de Blugeroo. Detectar vulnerabilidades en C y C++ con Flawfinder" [en línea]. Actualizada: 10 enero 2011. [Fecha de consulta: 07 abril 2011].Disponible en:http://blog.buguroo.com/?=http://backtrackworld.wordpress.com/2008/09/12/detectar-vulnerabilidades-en-c-y-c-con-flawfinder/.

4. "Atacando a Linux Herramientas para realizar auditorías". [en línea] [Fecha de consulta: Enero 2012]. Disponible en: <http://www.wikilearning.com/tutorial/atacando_linux-herramientas_para_realizar_auditorias/4250-2>.

5. "Ejemplo de auditoría automatizada: RATS" [en línea] [Fecha de consulta: Febrero 2012]. Disponible en:

<Http://www.debian.org/security/audit/examples/RATS>.

6. "Ejemplo de auditoría automatizada: pscan" [en línea] [Fecha de consulta: Febrero 2012]. Disponible en:
<http://www.debian.org/security/audit/examples/http://www.debian.org/security/audit/examples/pscan>.