Universidad de las Ciencias Informáticas

Facultad 1

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

“Herramienta de Auditoría de Código Fuente 1.1”

Autor: Annareya Mercedes Soler Rodríguez

Tutor: Ing. Michel Evaristo Febles Parker

Habana, junio de 2015


"NO VIVAS PARA QUE TU PRESENCIA SE NOTE, SINO PARA QUE TU AUSENCIA SE SIENTA"
-Bob Marley

Declaración de Autoría

Declaro que soy la única autora del trabajo titulado: Herramienta de Auditoría de Código Fuente 1.1 y se le otorga a la Universidad de la Ciencias Informáticas (UCI) los derechos patrimoniales del mismo, con carácter no exclusivo.

Para que así conste, firmo la presente a los ___ días del mes de junio del año 2015.

Firma del Autor

Annareya Mercedes Soler Rodríguez

Firma del Tutor

Ing. Michel Evaristo Febles Parker.

Agradecimientos

Que hoy sea ingeniera no es un logro al que se puede llegar sola, sin el apoyo de las personas más importantes de mi vida. Ha sido una gran carrera de mucho tiempo pero que valió la pena porque llegué a la meta.

Primero quiero agradecer a mis compañeros de aula, quienes estuvieron pendientes de mi recorrido, se interesaron por todo y fueron capaces de sorprenderme con solo una pregunta o con una palabra. A Ricard, Marlon, Daniel, Manu, Yile, Yisel, Neyvis, en fin, a todos, sepan que me llevo un buen recuerdo y que los extrañaré muchísimo. También a aquellos que conocí en el transcurso de la vida en la universidad: Sandra, Julito, Liadelis, Alexei, Yosvanis, Yoel Rojo, Rachel, Dairon, Leonardo, Migue, Raciel, Justiz y familia y a la mora Lili que contribuyó al cuidado de mi salud, así como a Yaiselin y Jose.

Mi agradecimiento a los que ya no están, pues la vida les proporcionó otro camino, pero aún así no se separaron de esta ardua carrera.

Agradecida estoy con todos los profesores que me han asistido en esta gran escuela de quienes me llevo un gran recuerdo, incluyendo a los profesores del departamento de sistema operativo de CESOL.

Le agradezco a mi tutor por estar presente en todo momento.

Un agradecimiento especial a aquellos que desde hace mucho tiempo son mis amigos y confidentes, que no dejan que la distancia nos separe, a Dánae y familia por su amor, por estar siempre en las buenas y malas, por ser sincera aunque muchas veces dolió, pero así son los amigos. A Zuleidys y familia por estar en todo momento, por brindarme su ayuda, amistad, cariño y amor. A Omar por acompañarme toda mi estancia en la escuela y soportarme. A Roxana por tenerme siempre presente y por tantas palabras alentadoras que me dio en su momento, al igual que Nani, María y familia, Gladys y familia, Dayana. A Edelsys y Miguelito por todo lo que me hicieron entender y comprender, por estar y si no, llamar y así estar al tanto.

Quiero agradecer a los amigos que hice ya en esta carrera, que de no ser por mi niño no los hubiese conocido, pues a pesar de todo estuvieron pendientes de mi carrera y de mi vida. Muchas gracias a Nina y familia, Yamilet y familia, Aliesky y familia, Gretter y familia, Dayersy y familia, también tengo que mencionar a mi niñita hermosa Daverly por los quieros y abrazos que me dio cuando menos me lo esperaba, ya que ello abrigó mucho mi corazón.

Son muchas las amistades que estuvieron a lo largo de esta gran carrera y no quiero pasar por alto a ninguno, pero a los que no menciono ahora sepan que les estoy muy agradecida, tan solo por brindarme una sonrisa cuando más lo necesitaba.

Agradecer a mi familia por su cariño, disposición, por sus palabras y risas.

Agradecer a ese niño de mis ojos y a quien amo más de lo que yo era capaz de imaginar, gracias mi amor por estar y saber cuando hacer y cuando no, gracias por todo cuanto eres. Agradecer a mi suegra por estar presente.

Sobretodo un agradecimiento eterno a mi madre por todo cuanto soy y cuanto seré, pues principalmente a ella le debo el resultado de esta gran carrera, porque sabía que en la meta estaba esperándome para un abrazo, gracias mami por velar por mí todo el tiempo, gracias por ser como eres y luchar contra viento y marea. Quiero que sepas que a ti te dedico este gran logro.

Dedicatoria

A mi madre por hacerme ver la vida de otro modo, por enseñarme todo lo que sé y hacer de mí la mujer que hoy soy, por estar siempre conmigo.

A mi abuela que donde quiera que se encuentre, sé que está orgullosa.

Resumen

En el Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI) actualmente se desarrolla la distribución cubana GNU/Linux Nova. La distribución cuenta con la herramienta de Auditoría de Código Fuente (ACF) versión 1.0 (v1.0) la cual actualmente no audita íntegramente el código fuente de todas las aplicaciones y bibliotecas que se encuentran en el repositorio de Nova, comprometiendo así la seguridad de la distribución. El presente trabajo va enmarcado a la necesidad de desarrollar la versión 1.1, con el objetivo de auditar todo el código fuente del repositorio de la distribución. Para el desarrollo de la solución se utilizaron las herramientas siguientes: *Visual Paradigm* para UML (*Unified Modeling Language* por sus siglas en inglés) versión 8.0, QtCreator versión 2.3, Doxygen versión 1.7.8. Como lenguaje de programación se utiliza C++. La metodología guía será *OpenUp*. Finalmente la herramienta de ACF v1.1, es capaz de auditar los 19 362 paquetes de código fuente que se encuentran en el repositorio de la distribución Nova, hacen posible la detección de las vulnerabilidades en todos los paquetes.

Palabras clave: auditoría, código fuente, Nova, repositorio.

Índice de contenido.

Índice de ilustraciones

Índice de tablas

Introducción

Francis Bacon, Barón de Verulam, en 1575 expresó: “La información es poder“ [1]. Es por ello que el acceso a la misma constituye una necesidad primordial, requiriendo una adecuada protección y aseguramiento. Para lograr que la información sea segura es necesario establecer un balance correcto entre los términos: confidencialidad, disponibilidad e integridad.

Hoy en día las formas de brindar información han proliferado, y esto compromete aún más la seguridad de ella. Una de las maneras de ofrecer información es a través de las Tecnologías de la Información y las Comunicaciones (TIC).

Cuba a pesar de no mantenerse a la par con el avance de las tecnologías que imponen los países capitalistas no se encuentra ajeno al desarrollo de ellas. Actualmente la seguridad de la información en el país se encuentra en riesgo debido a la dependencia absoluta de productos informáticos extranjeros que día a día juegan un papel cada vez más importante en las esferas gubernamentales, productivas y sociales de Cuba.

Las aplicaciones y herramientas fabricadas por cubanos y para los cubanos, deben adaptarse no solo al lugar en el que son implementadas, sino que deben extenderse más allá del medio en que son desarrolladas (socio-adaptabilidad). Así mismo, deben cubrir las necesidades que la nación posee, e incitar al desarrollo de forma autónoma en la esfera tecnológica (soberanía tecnológica). El estudio constante y las investigaciones que tributen al conocimiento de nuevas tecnologías y el uso provechoso de los recursos dispuestos para ello podrán garantizarle a las aplicaciones desarrolladas vigencia y que sean sostenibles por un buen tiempo (sostenibilidad). También deben garantizar un sistema libre de ataques y sin puertas traseras, a través de una correcta auditoría de código (seguridad) [2].

Desde febrero del 2009 el país cuenta con la distribución de GNU/Linux Nova, buscando incrementar niveles de sostenibilidad, socio-adaptabilidad, soberanía tecnológica y seguridad, haciendo énfasis en este último mencionado. La distribución se desarrolla en el departamento de Sistema Operativo de CESOL en la UCI.

Paralelo al surgimiento de la distribución de GNU/Linux Nova nace la herramienta de Auditoría de Código de Fuente (ACF) v1.0. Esta se utiliza para encontrar fallas de seguridad en el código fuente de una aplicación o bibliotecas que se encuentre desarrollada en los lenguajes C, C++, Bash, Perl y Python, así

como de la existencia de posibles ataques de carácter político. Esta herramienta tiene como objetivo fundamental auditar el código fuente de las aplicaciones y bibliotecas que residen en el repositorio[1] de Nova.

ACF v1.0 se ha utilizado satisfactoriamente para auditar paquetes de código fuente que contienen 50 ficheros de código fuente o menos, los que se pueden considerar como paquetes pequeños; en cambio para paquetes de mayor cantidad de archivos, el tiempo de ejecución es superior a 2 horas y en otros casos no concluye la revisión; siendo evidencia de ello una revisión realizada al paquete openssl-1.0.1. Dicho paquete tiene un tamaño de 20.5 MB[2] lo que equivale a más de 2000 elementos, una vez entrado al directorio se tiene un total de 20 carpetas y cada una de ellas posee varios subdirectorios. Actualmente la herramienta necesita una elevada cantidad de memoria para ejecutar sus procesos y no libera dicha memoria una vez terminada la realización de los mismos, por lo que al consumir tantos recursos el sistema operativo decide terminar con todas las aplicaciones en ejecución. Esto sucede al auditar directorios de gran tamaño como el de openssl-1.0.1. La herramienta en cuestión no analiza ficheros que no tienen extensión pero sí contienen código fuente. El repositorio de Nova posee 19 362 paquetes de código fuente, todos con un número variable de ficheros[3].

Por tanto, en estos momentos no se considera que la herramienta audite todos las aplicaciones y bibliotecas que residen en el repositorio de Nova, lo que conlleva a no poder detectar de forma automática y más temprana, las vulnerabilidades que pudiesen existir en el repositorio y que puedan comprometer la seguridad de la distribución de GNU/Linux Nova.

Por lo antes dicho se plantea como **problema a resolver** en esta investigación: ¿Cómo lograr que la herramienta ACF pueda auditar íntegramente el código fuente de todas las aplicaciones que se encuentran en el repositorio de la distribución de GNU/Linux Nova?

El **objeto de estudio** son las herramientas de auditoría de código fuente, y el **campo de acción** es la herramienta ACF.

Se plantea como **objetivo general** desarrollar la herramienta de ACF que audite íntegramente el código fuente de todas las aplicaciones que se encuentran en el repositorio de la distribución de GNU/Linux Nova.

1 Repositorio es el sitio donde se almacena información digital, desde archivos hasta programas.
2 MB es la abreviatura de megabytes.
3 Dato ofrecido por el administrador del repositorio de Nova Ing. Héctor Pérez Baranda, el 20 de abril del 2015.

Se determinan como **objetivos específicos:**

- Analizar las herramientas de auditoría de código fuentes existentes.
- Desarrollar la herramienta ACF v1.1.
- Probar que la herramienta ACF v1.1 audita todo el código fuente de todas las aplicaciones y bibliotecas que se encuentran en el repositorio de la distribución.

En el presente trabajo de diploma se plantea como **idea a defender** que con el desarrollo de la herramienta ACF v1.1 se podrá auditar íntegramente el código fuente de todas las aplicaciones que se encuentran en el repositorio de la distribución GNU/Linux Nova.

Para alcanzar el objetivo se han propuesto como **tareas a cumplimentar** las siguientes:

- Caracterización del estado del arte asociado a soluciones similares desarrolladas para sentar las bases en cuanto a las herramientas de auditoría de código fuente.
- Fundamentación de la metodología, del lenguaje y las tecnologías empleadas para el desarrollo de la solución.
- Definición y refinamiento de los requerimientos funcionales y no funcionales de la herramienta para el desarrollo de la propuesta de solución.
- Análisis de la arquitectura e implementación de ACF v1.1 para desarrollar la solución.
- Desarrollo de las funcionalidades requeridas para el progreso de la solución propuesta.
- Diseño y ejecución de casos de pruebas para comprobar la solución propuesta.

Para la revisión de la literatura y así dar cumplimiento al objetivo propuesto y a las tareas, se utilizan como **métodos de investigación científica** los siguientes:

Métodos teóricos utilizados:

- Inductivo-deductivo: Se utilizó para el estudio de las herramientas de auditoría de código fuente, atendiendo a sus características propias, así como su funcionamiento y basado en ello se definieron características que fueron eslabón importante en el desarrollo de la herramienta ACF v1.1.
- Analítico-sintético: Permitió realizar un estudio de la teoría y antecedentes relacionados con cada una de las herramientas de auditoría de código fuente, y de esta manera extraer las particularidades más importantes para la comprensión del funcionamiento de las mismas.

La estructura capitular del presente trabajo es la siguiente:

Capítulo 1: “Fundamentación teórica de las herramientas de auditoría de código fuente”: Este capítulo incluye un estado del arte de las herramientas de auditoría de código fuente, conceptos fundamentales necesarios para el entendimiento de la investigación, así como el estudio de las herramientas homólogas. También se deja definida la metodología y las tecnologías utilizadas para el desarrollo de la solución.

Capítulo 2: “Análisis, diseño e implementación de la herramienta ACF v1.1”: Este capítulo brinda una visión del sistema, se definen los requisitos funcionales y no funcionales a los que se debe dar cumplimiento, se ofrece además la propuesta del sistema a desarrollar. Se realiza el diseño y descripción de la arquitectura del sistema, y se presentan además diagramas de clases y de secuencia. También cuenta con los estándares de codificación utilizados para el desarrollo de la solución.

Capítulo 3: “Pruebas realizadas a la herramienta ACF v1.1”: Este capítulo describe las pruebas realizadas a la herramienta ACF v1.1. Se deja definido los métodos, las estrategias, las técnicas utilizadas, así como los casos de pruebas y las no conformidades encontradas en la solución propuesta.

Capítulo 1: Fundamentación teórica de las herramientas de auditoría de código fuente.

Introducción

En el presente capítulo se abordarán temáticas referentes a la actual herramienta ACF v1.0, así como elementos que sustentan la investigación. Es importante señalar que esta investigación no se realiza con el objetivo de seleccionar una herramienta existente para satisfacer las necesidades actuales, sino para tomar ideas y conocer los principales conceptos sobre este tema, esto se debe fundamentalmente a que el resultado del presente trabajo forma parte de una herramienta con características muy específicas.

1.2 Conceptos asociados a la investigación

Para lograr una mejor asimilación de los temas es necesario definir los conceptos que se exponen a continuación.

- *Repositorio:* Un repositorio o depósito es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos, o sea, es un lugar donde se almacena información [3]. Es el lugar donde se archiva toda la información, que pueden ser herramientas, aplicaciones, documentos digitales.
- *Fallas de seguridad:* Es el error que se encuentra en el sistema que no solamente puede ser en la sintáxis del código, sino también puertas traseras abiertas, desbordamiento de búffer[4], inyección de código SQL[5], asignación de permisos a usuarios a los que no le corresponden. Todo ello tributa contra la seguridad del sistema.
- *Código fuente:* Texto escrito en un lenguaje de programación específico y que puede ser leído por un programador. Debe traducirse a lenguaje máquina para que pueda ser ejecutado por la computadora o a *bytecode*[6] para que pueda ser ejecutado por un intérprete. Este proceso se

4 Se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (búffer).
5 Errores de programación.
6 Entiéndase por bytecode el código intermedio más abstracto que el código máquina [5].

denomina compilación [4].

- *Auditoría:* 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 [6].

Una auditoría de código fuente es un proceso de *hacking* ético, a través del cual se ejecutará un plan completo de pruebas para identificar las posibles vulnerabilidades existentes en el código fuente de una aplicación. Requiere un componente manual capaz de asimilar el contexto de la aplicación para obtener resultados óptimos.

1.3 Estudio de las herramientas de auditoría de código fuente

Realizar auditorías manualmente es un proceso arduo. A pesar de ser una de las soluciones más efectivas, este proceso requiere de grandes costes debido a la magnitud del código. También el tiempo de espera es prolongado y es por ello que a menudo se explota el producto sin esperar resultados. La realización de estas auditorías depende de un desarrollo terminado.

La auditoría de código de una aplicación forma parte del ciclo de vida en el desarrollo de software y fomenta el uso de buenas prácticas. Mejora la calidad y minimiza el mantenimiento de código.

El creciente desarrollo de las TIC ha permitido el desenlace de herramientas que admiten la detección de vulnerabilidades en el código de las aplicaciones, concediendo la localización de posibles fallas de seguridad.

Existen muchas herramientas para examinar programas o*pen source*[7] y muchas están diseñadas para encontrar fallas de seguridad en aplicaciones escritas en C, ya que la mayoría de los mismas se desarrollan en este lenguaje.

A continuación se muestra el análisis realizado a varias herramientas.

Buguroo | BugScout

Esta herramienta de licencia privativa es un analizador de código que posee evasión de falsos positivos. Es una herramienta diseñada para automatizar el análisis estático del código fuente de aplicaciones web.

El objetivo principal de BugScout es determinar el nivel de seguridad del aplicativo y proponer acciones correctivas o mitigadoras para las vulnerabilidades encontradas.

7 Open Source o Código abierto es la expresión con la que se conoce al software distribuido y desarrollado libremente.

Consta de una consola web desde la que se ofrecen múltiples funcionalidades para operar con el código, sin necesitad de pesados agentes ni software en cliente. Permite con facilidad gestionar las vulnerabilidades, realizar informes, almacenar documentación, consultar estadísticas [7].

Dicha herramienta posee un sistema de detección de vulnerabilidades públicas y privadas actualizado al día. Tiene plataforma multiauditoría, capaz de analizar códigos de manera simultánea sin interferir en el rendimiento y tiempos de la misma [7].

La herramienta analiza código fuente de varios lenguajes (PHP, Java, HTML, XML, JSP, ASP, ASPX, Visual Basic.NET, C#), e indexa las vulnerabilidades detectadas de manera jerárquica gracias al empleo de una taxonomía abierta.

La herramienta proporciona las siguientes ventajas:

- Motor de reglas personalizable para los clientes que requieren un alto nivel de personalización.
- Permite reducir el número de falsos positivos.
- Evita errores no controlados: denegación de servicio, ataques de *spam*.
- Permite actualizar firmas de carácter público y privado a tiempo real.
- Se integra con el desarrollo del ciclo de vida de software.
- Facilita el cumplimiento normativo o la adaptación a los requerimientos específicos de cada empresa.
- Se adapta con rapidez a las nuevas vulnerabilidades que aparecen en el campo de la seguridad.
- El núcleo de BugScout tiene una capacidad de análisis de hasta 300.000 líneas por minuto. Podría auditar el código de Facebook en menos de media hora [8].

Destacar que la herramienta detecta alrededor de 5575 casuísticas distintas de vulnerabilidades (más del 94% de las vulnerabilidades presentes en el código), funciona en *cloud*, permitiendo escalabilidad ilimitada. BugScout admite políticas de detección personalizables en base a una taxonomía de vulnerabilidades propia [8].

ITS4

ITS4 es una herramienta que pertenece a la sección *non-free* (no libre) del archivo de Debian, y sólo está disponible para la distribución. Dicha herramienta se puede usar para buscar potenciales agujeros de seguridad en C y C++. La salida que produce no tiene en cuenta algunos de los casos en los que las

llamadas a las funciones peligrosas se han hecho con cuidado [9].

Tiger

Tiger es un conjunto de *scripts* que escanean el sistema en busca de problemas de seguridad, ello incluye una búsqueda en el código fuente. Se puede utilizar tanto como una auditoría de seguridad como para detección de intrusos. Es compatible con múltiples plataformas UNIX, es gratis y siempre bajo una licencia *General Public License* (GPL). A diferencia de otras herramientas, Tiger sólo necesita de herramientas POSIX[8] y está escrito completamente en lenguaje *shell*. Al ejecutar Tiger, este empezará a lanzar una serie de diagnósticos planos muy completos, que informarán de la situación de cuentas, ficheros y servicios, comprobando permisos en usuarios, contraseñas, grupos, configuraciones de servicios y procesos [10].

Tiene un Tigercron[9] para programar revisiones periódicas, con esta herramienta sería fácil al crear el fichero hacer un *script* que envíe una alerta al usuario si no coincide con una plantilla que hayamos creado y notifique lo que se diferencia [10].

Lynis

Lynis es una herramienta de seguridad y auditoría de sistemas *open source* muy potente y personalizable que permite tener una visión clara del estado del sistema en cuanto a vulnerabilidades, *hardening*[10] y seguridad en general.

Dicha herramienta realiza una gran cantidad de pruebas contra el sistema para determinar el estado de la seguridad de este, por ejemplo, la búsqueda de software instalado y sus vulnerabilidades o la revisión de configuración en busca de fallas de seguridad y software desactualizado. Todo ello queda reflejado en un informe final junto con un listado de recomendaciones para solventar todos los fallos encontrados. Así pues, esta herramienta es de utilidad para realizar auditorías de sistemas, escaneo de vulnerabilidades y *hardening* del sistema [11].

Se puede hacer un escaneo rápido o completo, definir únicamente ciertos *tests*, modificar la apariencia del

8 POSIX es la abreviación de *Portable Operating System Interface*. Se trata de un estándar que intenta asegurar la portabilidad entre diferentes sistemas operativos.

9 Tigercron se utiliza para ejecutar periódicamente cheques de la herramienta Tiger.

10 Es el proceso de asegurar un sistema mediante la reducción de vulnerabilidades en el mismo, esto se logra eliminando software, servicios, usuarios, etc.

informe, revisar actualizaciones, ejecutar en modo *debug*, y añadir más *plugins* (para aumentar las funcionalidades de la herramienta o subir el informe a un sistema externo) [12].

La herramienta analiza todo el código fuente del sistema (servicios inseguros, software, herramientas del sistema).

Hammurapi

Esta es una herramienta libre que analiza más de 100 deficiencias en el código fuente Java, entre ellas la conformidad con la especificación de los EJBs[11], problemas con los *threads*[12], y otros muchos si el código sigue los estándares de codificación Java. Todas estas deficiencias poseen un nivel de gravedad, que se emplea en los informes generados por la herramienta. La misma presenta primero la información más importante. Hammurapi está enfocada al análisis de aplicaciones grandes (varios *megabytes* de código fuente) y a reducir el mantenimiento de software. Ofrece un *plugin* para Eclipse y de esta manera simplificar su uso [13].

RATS

Rouge Auditing Tool for Security, es una herramienta que permite verificar códigos escritos en C, C++, Perl, Python y PHP. Este programa nos alertará cuando cometamos algún error común al programar. Es bastante útil como primer paso para realizar una revisión del código, pero no es un sustituto completo de esta revisión [14].

Esta herramienta proporciona un análisis de seguridad con una lista de posibles puntos débiles sobre los que se centran, además de la descripción del problema, y potencialmente sugieren soluciones. También proporciona una evaluación relativa de la gravedad potencial de cada problema, para ayudar mejor a un auditor a priorizar. Para cada vulnerabilidad encontrada imprime una explicación del problema (si está disponible en la base de datos de vulnerabilidades). Finalmente RATS imprime el número de las líneas analizadas y el tiempo que utilizó [15].

Esta herramienta también realiza algunos análisis básicos para tratar de descartar trastornos que, obviamente, no son problemas. Como su nombre lo indica, la herramienta sólo realiza un análisis aproximado de código fuente. No va a encontrar todos los errores y también encontrará elementos que no

11 Enterprise JavaBeans (también conocidos por sus siglas EJB).

12 Threads o hilos de ejecución es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

son errores.

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.

Características:

- Reduce los riesgos empresariales mediante la detección, la priorización y la solución de vulnerabilidades que suponen la mayor amenaza.
- Identifica y elimina vulnerabilidades que pueden explotar rápidamente con un proceso repetitivo.
- Reduce los costes de desarrollo identificando las vulnerabilidades antes en el SDLC[13].
- Educa a los desarrolladores en prácticas de codificación segura mientras trabajan.
- Reúne a los equipos de desarrollo y seguridad para encontrar y solucionar los problemas de seguridad [16].

Flawfinder

Esta es una herramienta diseñada para analizar código fuente C y C++ en busca de potenciales fallas de seguridad en todo el sistema. Este programa busca funciones con problemas conocidos, como desbordamientos de búffer, errores de formato, condiciones de carrera, posibles problemas de uso de metacaracteres en la shell[14] y generadores de números aleatorios. La aplicación emitirá un informe con todos los posibles errores, avisando de posibles problemas de seguridad.[17] Estos problemas son ordenados por riesgo, que sería un número del 1 al 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 [18].

El programa funciona escaneando el código y buscando el uso de las funciones que tiene en su base de datos que normalmente se usan mal.

Para facilitar la lectura del informe, se puede indicar que contenga la línea en la que se usa la función, lo

13 "*System Design Life Cycle*" (ciclo vital del desarrollo/diseño de sistemas).

14 La shell, intérprete de ordenadores o concha se emplea para referirse a aquellos programas que proveen una interfaz de usuario para acceder a los servicios del sistema operativo. Metacaracteres de la shell son los caracteres(* ? $ | & 2> 2>> < &&) que tienen un significado propio para la shell.

que puede ser útil para detectar un problema inmediatamente.

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 *variadic*[15] en código fuente C y C++, como printf, fprintf y syslog [18].

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 se pueden descubrir y reparar.

1.4 Comparación de las herramientas de auditoría de código fuente

A continuación se realiza la comparación de las herramientas valorando los aspectos que son necesarios para lograr una mejor compresión de la presente investigación. Se incluye la herramienta ACF v1.0.

Herramientas	Plataforma	Licencia	Lenguajes de programación	Tipo de vulnerabilidades	Interfaz gráfica
Buguroo BugScout	Windows	Privativa	PHP, Java, NET, HTML, XML, JSP, ASP, ASPX, Visual Basic.NET, C#.	*Open redirect*[16], pérdida de memoria, agujeros de seguridad.	Sí
ITS4	Debian	Libre	C, C++.	Agujeros de seguridad.	No
Tiger	Debian	Libre	Shell.	Pérdida de memoria, agujeros de seguridad.	No
Lynis	Linux u otro sistema UNIX.	Libre	C.	*Phishing*[17], *malware*[18], errores de programación y de configuración.	No
LCLint	Debian	Libre	C.	Localiza posibles violaciones de información oculta, modificaciones inconsistentes de estados, errores en la gestión de	No

15 Funciones que no siempre toman el mismo número de argumentos.
16 *Open redirect* se trata de una vulnerabilidad que permite al atacante realizar una redirección arbitraria.
17 *Phishing* o suplantación de identidad.
18 *Malware*: el objetivo es infiltrarse y dañar la computadora sin el consentimiento del propietario.

Herramientas	Plataforma	Licencia	Lenguajes de programación	Tipo de vulnerabilidades	Interfaz gráfica
				memoria, uso de memorias no definidas, no referencias al puntero nulo, usos de alias inesperados.	
RATS	Debian	Libre	C, C++, Perl, Python, PHP.	División por cero, ciclo infinito, problemas aritméticos como desbordamientos, exceder el tamaño del arreglo, utilizar una variable no inicializada, acceder a memoria no permitida, pérdida de memoria, desbordamiento o subdesbordamiento de la pila, desbordamiento de búffer, bloque mutuo.	No
HP Fortify SCA	Linux, Oracle SDolaris	Libre	HTML, Java, PHP, Python, JSP, XML, JavaScript/Ajax, C, C++.	Asignación a variables miembro en *servlets* Java, identifica el uso de registradores que no están declarados *static* final e instancias banderas de código muerto que nunca serán ejecutadas por un predicado que es siempre falso, errores, debilidades y violaciones de política en los archivos de configuración de despliegue de una aplicación, desbordamiento de búffer que implican escribir o leer más datos que un tampón puede contener.	No
Flawfinder	Debian	Libre	C, C++.	Desbordamiento de búffer, errores de formato, condiciones de carrera, generadores de números aleatorios.	No
Pscan	Debian	Libre	C, C++.	Errores de cadena de formato.	No
ACF v1.0	Debian	Libre	C, C++, Perl, Python, Bash.	Desbordamiento de búffer, condición de carrera, denegación	Sí

Herramientas	Plataforma	Licencia	Lenguajes de programación	Tipo de vulnerabilidades	Interfaz gráfica
				de servicio, escalada de privilegios, identifica posibles ataques de carácter político.	

Tabla 1: Comparación de las herramientas de auditoría de código fuente (elaboración propia).

1.5 Valoración de las herramientas de auditoría de código fuente estudiadas

Luego de un estudio de las diversas herramientas de auditoría de código fuente se concluye que:

- Al identificar las características más prominentes de las herramientas de auditoría de código fuente, se pudo constatar que la más completa es propietaria y de las restantes, una por sí sola no reúne todas las funcionalidades deseables en la herramienta que se quiere complementar como resultado de la presente investigación.
- Las herramientas cuentan con funcionalidades particulares que la diferencian de las demás, y en ocasiones son complementarias siendo de gran utilidad si se encuentran todas reunidas en una sola solución.
- A pesar de lo antes expuesto las herramientas auditan en base al propósito bajo el cual fueron implementadas.
- Las herramientas libres presentadas analizan las aplicaciones que posee el sistema pero no son capaces de detectar vulnerabilidades en un repositorio de aplicaciones y bibliotecas.
- La herramienta ACF v1.0 debe escanear el repositorio de Nova 2013, para detectar vulnerabilidades en todos los ficheros de código fuente.

1.6 Metodología y tecnologías empleadas

La calidad de un producto de software es determinada en gran medida por la calidad del proceso utilizado para desarrollarlo y mantenerlo, por lo que se dedica esta sección a la selección de la metodología de desarrollo, del lenguaje de programación, así como las herramientas a utilizar en el desenlace del presente trabajo.

1.6.1 Metodología de desarrollo de software

Un proceso de software detallado y completo suele denominarse metodología, define quién debe hacer, qué, cuándo y cómo debe hacerlo. Las metodologías son muy necesarias a la hora de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas engloban procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software.

Una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un producto de software [19].

Estas constituyen la guía para lograr un proceso disciplinado en el ciclo de desarrollo del software para de esta manera cumplir con el objetivo, haciéndolo eficiente y predecible.

OpenUp

Es un proceso de desarrollo unificado que está basado en *Rational Unified Process* (RUP[19]) y forma parte de la familia del proceso unificado que lo conforman además de los mencionados *Agile Unified Process* (AUP[20]), *Enterprise Unified Process* (EUP[21]), *Basic Unified Process* (BUP[22]) [20].

Esta metodología es un proceso mínimo y suficiente, lo que significa que solo el contenido fundamental y necesario es incluido. Por lo tanto no provee lineamientos para todos los elementos que se manejan en un proyecto pero tiene los componentes básicos que pueden servir de base a procesos específicos y la mayor parte de los elementos de esta metodología están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances. El ciclo de vida de un proyecto, según *OpenUP*, permite que los integrantes del equipo de desarrollo aporten con micro-incrementos, que pueden ser el resultado del trabajo de unas pocas horas o unos pocos días. El progreso se puede visualizar diariamente, ya que la aplicación va evolucionando en función de estos micro-incrementos. El objetivo de *OpenUP* es ayudar al equipo de desarrollo, a lo largo de todo el ciclo de vida de las iteraciones, para que sea capaz de añadir valor de negocio a los clientes, de una forma predecible, con la entrega de un software operativo y funcional al final

19 Proceso de unificado racional.

20 El proceso de unificado ágil (*Agile Unified Process* por sus siglas en inglés) es la versión simplificada de RUP.

21 Proceso de unificado empresarial.

22 Proceso de unificado básico.

de cada iteración. [21].

OpenUP en cada iteración del ciclo de vida, incrementa progresivamente los objetivos de las iteraciones anteriores y añade nuevas funcionalidades a las versiones estables del software que se tiene hasta el momento [22].

De manera general, *OpenUP* es un proceso de desarrollo de software de código abierto que aplica propuestas iterativas e incrementales dentro del ciclo de vida, tratando de ser manejable en relación con el RUP. Se valora la colaboración y el aporte de los *stakeholders*[23] sobre los entregables y la formalidad innecesarios.

Después de realizar un análisis de la metodología se seleccionó *OpenUP* por las siguientes características:

- Es una de las metodologías utilizadas por el Departamento CESOL.
- Es la que más se acoge a las necesidades del equipo de desarrollo.
- Es una metodología ágil de desarrollo del software.
- Es extensible ya que en el proceso se pueda agregar o adaptar según lo vayan requiriendo los sistemas.
- Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.

1.6.2 Herramienta CASE

Con el desarrollo de las tecnologías existen actualmente más herramientas que aumentan la productividad y eficiencia del desarrollo de software, de esta forma se minimiza el costo en términos de tiempo y dinero. Estas son las llamadas herramientas CASE (del inglés *Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computador) [23].

Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de

23 La definición más correcta de "*stakeholder*" sería parte interesada (del inglés *stake*, apuesta, y *holder*, poseedor). Se puede definir como cualquier persona o entidad que es afectada o concernida por las actividades o la marcha de una organización.[9]

errores.

En la actualidad existen un gran número de herramientas CASE. *Visual Paradigm* es una herramienta muy poderosa y profesional que soporta el ciclo de vida del desarrollo de un software. Tiene características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar UML, que son: diagramas de clase, casos de uso, comunicación, secuencia, estado, actividad, y componentes [23].

Visual Paradigm para UMLversión 8.0

UML es un lenguaje de modelado gráfico que propicia organizar, construir y documentar los elementos que forman un sistema de software. Permite modelar artefactos conceptuales como lo son procesos de negocio y funciones de sistema, además de artefactos concretos estos como escribir clases en un lenguaje determinado, esquemas de bases de datos y componentes de software reusables. UML es un lenguaje para especificar y no un método o un proceso. Aunque es flexible y permite aplicarse en una gran variedad de metodologías de desarrollo, no especifica en sí cual se debe utilizar. Para modelar, UML utiliza diagramas que se pueden clasificar en dos tipos: diagramas de estructura que comprenden los diagrama de clases, diagrama de componentes, diagrama de objetos, diagrama de despliegue y diagrama de paquetes; y también en diagramas de comportamientos entre los que se encuentran diagrama de actividades, diagramas de casos de uso, diagramas de estados, diagrama de secuencia y diagrama de colaboración. [23].

Visual Paradigm es una herramienta CASE que ha sido concebida para soportar el ciclo de vida completo (análisis y diseño orientados a objetos, construcción, pruebas y despliegue) de una aplicación a través de diagramas. La misma propicia un conjunto de ayudas para el desarrollo de programas computacionales, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

Características:

- Soporta las últimas versiones del UML y Notación de Modelado del Proceso de Negocio[24] (BPMN por sus siglas en inglés).
- Disponibilidad en múltiples plataformas tanto en sistemas operativos Windows como en las distribuciones de GNU\Linux.

24 Es una notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo.

- Además del soporte de modelado UML esta herramienta provee el modelado de procesos de negocios, además de un generador de mapeo de objetos-relacionales para los lenguajes de programación Java.NET y PHP.
- Se integra con las herramientas Java: *Eclipse/IBM WebSphere, Jbuilder, NetBeans IDE, Oracle Jdeveloper, BEA Weblogic* [24].

Visual Paradigm para UML es apoyado por un conjunto de lenguajes de programación, tanto en la generación del código como en la ingeniería inversa, tales pueden ser: Java, C++, CORBA IDL, PHP, XML *Schema*, Ada y Python. Dicha herramienta apoya la generación del código C#, VB. NET, *Object Definition Language* (ODL), *Flash ActionScript, Delphi, Perl, C-Objetivo, y Ruby* [24].

Se utiliza *Visual Paradigm* en su versión 8.0 porque:

- Cumple con el *Standard* UML 2.0 y 2.1, lo que permite una Notación de Modelado del Proceso de Negocio.
- Posibilita modelar todo el ciclo de desarrollo del software y es multiplataforma.
- Posee capacidades de ingeniería directa e inversa.
- Modelo y código permanecen sincronizados durante todo el ciclo de desarrollo.
- Posee gran disponibilidad de integración con varios entornos de desarrollos integrados que están bajo licencias de software libre.
- Es sencilla de instalar, fácil de utilizar y actualizar.
- Es una suite completa de herramientas CASE.
- Es la herramienta que utiliza el centro CESOL en sus producciones.

Para el desarrollo de la presente investigación, se utiliza la herramienta en cuestión por todas las características antes expuestas, pues a pesar de ser un software propietario, la universidad cuenta con su licencia.

1.6.3 Entorno de desarrollo Integrado (IDE)

Entiéndase por entorno de desarrollo integrado (IDE por sus siglas en inglés de *Integrated Development Environment*) a la aplicación de software, que provee habilidades comprensivas para facilitar al

desarrollador el desarrollo de software.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario (GUI). Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes [25].

QtCreator versión 2.3

QtCreator es un IDE multiplataforma que se ajusta a las necesidades de los desarrolladores que utilizan las bibliotecas de Qt. Posee editor de código con soporte para C++, QML y *ECMAscript*, control estático de código y estilo a medida que se escribe. Es usado para diseñar interfaces gráficas de usuario [26].
Dicho IDE permite desarrollar aplicaciones en C++ de manera sencilla y rápida pues posee un avanzado editor de código C++. Es basado en la librería QT y posee herramientas para la administración y construcción de proyectos, depurador visual, interfaz gráfica de usuario integrada, diseñador de formularios y auto-completado de código. El objetivo principal de Qt Creator es conocer las necesidades de desarrollo de los programadores Qt que buscan la simplicidad, facilidad de uso, productividad, extensibilidad y apertura.
Las características claves de Qt Creator permiten a los programadores realizar las siguientes tareas:

- Empezar a desarrollar aplicaciones con Qt de manera rápida y fácil a través del asistente de proyectos, así como acceder rápidamente a proyectos recientes y sesiones.
- Diseñar aplicaciones de interfaz de usuario basadas en *widgets*[25] Qt con el editor integrado, Qt Designer.
- Desarrollar aplicaciones con el editor de código avanzado de C++, que provee nuevas y poderosas características para completar recortes de código, remanufactura de código, y ver el esquema de archivos (que es la jerarquía de símbolos de un archivo).
- Compilar, ejecutar y desarrollar proyectos Qt que se dirigen a múltiples plataformas de escritorio y móviles, como Microsoft Windows, Mac OS X, Linux, Symbian, MeeGo, y Maemo.
- Depurar con el depurador GNU usando una interfaz gráfica de usuario con una mayor conciencia

25 *Widget* (o control) es un elemento de una interfaz gráfica de usuario que muestra información con la cual el usuario puede interactuar. Por ejemplo: ventanas, cajas de texto, *checkbox*, *listbox*.

de las estructuras de clases Qt.

- Usar herramientas de análisis de código para verificar la administración de memoria en sus aplicaciones.
- Desarrollar aplicaciones para dispositivos móviles y crear paquetes de instalación para dispositivos Symbian, MeeGo, y Maemo que pueden ser publicadas.
- Acceder fácilmente a información con el módulo del sistema de ayuda contextual de Qt [26].

El paquete de instalación de Qt Creator está disponible para Microsoft Windows, Mac OS X, y Linux. Qt Creator puede ser ejecutado en otras plataformas, pero requiere la compilación del código fuente disponible públicamente.

ACF v1.0 constituye una aplicación de QT, está programada en C++ y las interfaces están desarrolladas utilizando las bibliotecas de Qt, por lo cual para el desarrollo de la presente investigación, se utilizará el IDE de desarrollo QtCreator por todas las características antes expuestas, lo cual permite mantener la compatibilidad con las tecnologías utilizadas y de esta manera poder desarrollar satisfactoriamentela herramienta ACF v1.1.

1.6.4 Herramienta de documentación de código fuente

Doxygen versión 1.7.8

Doxygen es una herramienta para convertir los comentarios de un programa escrito en C, C++, VHDL, PHP, C#, y Java, en documentación suficientemente prolija y presentable como para poder publicarse. Maneja la salida de documentación en formatos: html, Latex, pdf, rtf. Lo más interesante de la presente herramienta es que la documentación se genera de manera automática, y a partir de los archivos fuente de modo que siempre será consistente con estos [27].

Esta herramienta funciona tanto en los sistemas Unix como en Windows.

Ventajas:

- Al escribir la documentación en el mismo archivo fuente en forma de comentarios, exime al usuario de mantener otra documentación separada del código fuentes.
- La documentación se genera en forma automática agregando diagramas de interacción entre los

diferentes módulos.

- Permite documentar variables, estructuras de datos, además de las funciones [27].

Las etiquetas más comunes que usa Doxygen y que se presentan en la herramienta ACF v1.1 son:

brief: Indica que lo que sigue a continuación es el resumen de la función/clase/estructura/enumerado documentado.

param: Parámetro de una función. Debe estar seguido del nombre de la variable a la que nos referimos [28].

Para el desarrollo de la presente investigación, se utiliza la herramienta Doxygen por todas las características antes expuestas, y porque es la utilizada en la herramienta ACF v1.0.

1.6.5 Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. Al ser interpretado puede resultar en un programa, página web o aplicación [29].

C++

C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Actualmente existe un estándar, denominado ISO[26] C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos [30].

Características:

- **Programación orientada a objetos:** La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Permite la reutilización del código de una manera más lógica y productiva para el uso de plantillas.
- **Portabilidad:** Un código escrito en ANSI C++ puede ser compilado en casi todo tipo de ordenadores y sistemas operativos sin hacer apenas cambios, siempre que exista el compilador.
- **Brevedad:** El código escrito en C++ es muy corto en comparación con otros lenguajes, sobretodo

26 Organización Internacional de Normalización o ISO.

porque en este lenguaje es preferible el uso de caracteres especiales como las "palabras clave".

- **Programación modular:** Un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Esta característica permite unir código en C++ con código en otros lenguajes de programación como ensamblador o el propio C.
- **Velocidad:** El código resultante de una compilación en C++ es muy eficiente, gracias a su capacidad de actuar como lenguaje de alto y bajo nivel. Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:
  - Posibilidad de redefinir los operadores (sobrecarga de operadores).
  - Identificación de tipos en tiempo de ejecución (por sus siglas en inglés RTTI) [30].

C++ es un lenguaje muy potente, debido a que permite trabajar tanto a alto como a bajo nivel; sin embargo, es a su vez uno de los que menos atrae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta mucho su aprendizaje.

Se escoge el lenguaje C++ porque:

- Da la posibilidad de gestionar memoria a través de punteros.
- Le permite al programador a través de la programación orientada a objetos diseñar aplicaciones desde el punto de vista más como una comunicación entre los objetos que como una secuencia estructurada de código.
- Al ser un lenguaje compilado, su velocidad de ejecución es más rápida que la de otros lenguajes como por ejemplo Java, que utiliza una máquina virtual para poder ejecutarse.
- La posibilidad de poder ser compilado en varias computadoras, con pocos cambios (portabilidad).

Los principales aspectos de C++ son: abstracción (encapsulación), soporte para programación orientada a objetos (polimorfismo) y soporte de plantillas o programación genérica (*templates*) [30].

Para el desarrollo de la presente investigación, se utiliza el lenguaje de programación C++, porque fue utilizado para desarrollar la herramienta inicial, en el sistema operativo Nova 2013 y por las características presentadas.

Conclusiones del capítulo

A partir del análisis del estado del arte y de la importancia de la seguridad de la distribución GNU/Linux Nova se decidió implementar una solución que facilite la detección de vulnerabilidades en todo el código fuente de los lenguajes C, C++, Bash, Perl y Python.

El presente capítulo permitió concluir que:

- ✔ Las herramientas de auditoría de código fuente estudiadas presentaron características y funcionalidades similares a la que se pretende desarrollar con el presente trabajo; sin embargo, no todas detectaron vulnerabilidades en los lenguajes C, C++, Perl, Python y Bash y no identificaron posibles ataques de carácter político.
- ✔ Todas las herramientas homólogas realizan la función para las que fueron desarrolladas, pero ninguna por sí sola detectó las vulnerabilidades en los lenguajes C, C++, Perl, Python y Bash.
- ✔ Las herramientas estudiadas sentaron las bases para el desarrollo de la herramienta ACF v1.1.
- ✔ Para el progreso de la herramienta se utilizará la metodología de desarrollo *OpenUp*, como lenguaje de programación C++ y como herramientas QtCreator, Doxygen y *Visual Paradigm* para UML.

Capítulo 2: Análisis, diseño e implementación de la herramienta ACF v1.1

Introducción

En el desarrollo de software es de gran importancia definir los procesos que intervienen, con el objetivo de lograr un mayor entendimiento del sistema a desarrollar, por parte de los clientes y desarrolladores.

El presente capítulo tiene como propósito exponer el contexto en que se desarrolla el problema que da origen a esta investigación. Para ello se identifican los requisitos funcionales, no funcionales y los casos de uso (CU) críticos que guiarán el desarrollo del sistema propuesto. Para detallar las actividades antes mencionadas, se realiza el diagrama de CU con el objetivo de mostrar las relaciones entre el actor del sistema y las funcionalidades en las que están involucrados, así como los diferentes tipos de diagramas que presentan la lógica y funcionamiento del sistema. Además se muestra la arquitectura utilizada, y aspectos que conciernen a la implementación de la solución.

2.1 Descripción del negocio

La herramienta ACF v1.0 se desarrolló con el objetivo de auditar aplicaciones de código fuente.

La herramienta tiene dos tipos de *plugins*: auditoría y descompresión, los cuales se encargarán de los lenguajes que se van a analizar (C, C++, Perl, Python y Bash) y de la descompresión de las carpetas que poseen extensión tipo tar.gz, rar y zip. Inicialmente la herramienta antes de acceder al repositorio que se va a auditar carga dichos *plugins*. Luego va a utilizar los datos que el usuario le especifica que son: dirección del directorio de código fuente que se desea auditar y dirección donde se generarán los reportes como resultado de la auditoría para empezar el análisis.

A continuación ACF va al directorio que se va a analizar y revisará fichero por fichero. En caso de encontrar algún fichero comprimido, lo descomprime y sigue buscando en los archivos internos ficheros que pueda analizar.

El análisis realizado será archivado en los reportes como resultado de la auditoría y estos van a ser almacenados en la dirección especificada por el usuario. Dichos ficheros que genera la herramienta contienen las vulnerabilidades que encontró la misma. Los ficheros se identifican por el nombre

ScanDetails.txt, el cual contiene toda la información detallada y ScanSumarize.txt, el cual contiene un resumen del resultado de la auditoría.

ACF v1.0 no solo permite auditar, sino que le permite al usuario:

- Listar toda la información de un directorio ya auditado, la cual se encuentra en la base de datos de la herramienta.
- Visualizar la información de un directorio específico a partir de parámetros introducidos por el cliente (fecha, hora).
- Eliminar la información de un repositorio, la cual se encuentra en la base de datos.
- Editar la información almacenada con respecto a un directorio de código fuente, dando la posibilidad de ponerle o cambiarle el nombre al fichero y el nombre de la persona que lo adicionó.
- Agregar a la base de datos información del directorio de código fuente a analizar.
- Graficar las incidencias y vulnerabilidades encontradas en el repositorio auditado.

Actualmente la herramienta ACF v1.0 no audita un directorio que posea más de 10 elementos de código fuente.

2.2 Descripción de la propuesta de solución

La presente investigación está enfocada a la corrección de la herramienta de Auditoría de Código Fuente ACF v1.0.

Realizar la corrección de la herramienta tiene como propósito fundamental:

- Eliminar el exceso de fuga de memoria durante la ejecución de la auditoría, esto permite que se analice todo el repositorio de código fuente.
- Implementar funcionalidades para que la misma analice todo fichero de código fuente en los lenguajes C, C++, Perl, Python y Bash, es decir, que ACF sea capaz de auditar ficheros sin extensión, pero que aún así poseen código en dichos lenguajes, estos ficheros son conocidos como *scripts*.
- Reutilizar clases y objetos en forma de componentes.
- Preparar las bases tecnológicas para incorporar los componentes que van a auditar los lenguajes

PHP, Java y JavaScript.

2.3 Requerimientos del sistema

Los requisitos constituyen el eslabón entre la definición de la aplicación y el diseño de la misma. A través del análisis de las necesidades de los sistemas, la evaluación de la factibilidad y la validación de especificaciones se logra una mejor comprensión del problema en cuestión.

Los requisitos pueden ser clasificados como funcionales y no funcionales. Se le denomina requisito funcional al que describe qué debe hacer el sistema. Describe las funciones que el software ejecuta [31].

Los requisitos no funcionales son restricciones en el servicio o función ofrecida por el sistema. Incluyen restricciones de tiempo, del proceso de desarrollo y estándares. Especifican propiedades del sistema, como confiabilidad y seguridad [31].

2.3.1 Requisitos funcionales

A continuación se describen los requisitos funcionales de la herramienta ACF v1.1.

RF1: Detectar vulnerabilidades de todos los ficheros de código fuente con extensión.

RF2: Detectar vulnerabilidades en *scripts*.

2.3.2 Requisitos no funcionales

A continuación se describen los requisitos no funcionales de la herramienta ACF v1.1.

- Requerimientos de software.

RNF1: La herramienta deberá ejecutarse sobre la distribución GNU/Linux Nova 2013.

- Requerimientos de hardware.

RNF2: Un ordenador que ejecute la aplicación contará con:

- ✓ Mínimo 1Gb de memoria RAM.
- ✓ Mínimo 7Gb de disco duro.
- ✓ Microprocesador Dual-core o superior.

- Restricciones de diseño.

RNF3: Emplear QtCreator 4.

RNF4: Estándar de programación del lenguaje C++.

- Disponibilidad.

RNF5: La herramienta debe estar disponible en el repositorio.

- Ayuda.

RNF6: La herramienta debe contar con un manual de usuario que está disponible en la opción “Ayuda” de la interfaz visual, para que de esta manera el usuario adquiera una mayor experiencia con el trabajo en la misma y logre un correcto uso de sus funcionalidades.

- Requerimientos de licencia.

RF7: Emplear licencia GPL.

2.4 Modelo de Casos de Uso del Sistema

2.4.1 Descripción de los actores del sistema

Para que un sistema cumpla con su principal objetivo tiene que existir un usuario que lo ponga en marcha y lo use a conveniencia, los actores. Los actores representan el ser externo que interactúa con el sistema.

Actor	Descripción
Mantenedor de paquete.	Persona que desea auditar ficheros de código fuente, y para ello interactuará con la herramienta ejecutando las funcionalidades que esta brinda.

Tabla 2: Actores del sistema.

2.4.2 Diagrama de casos de uso del sistema

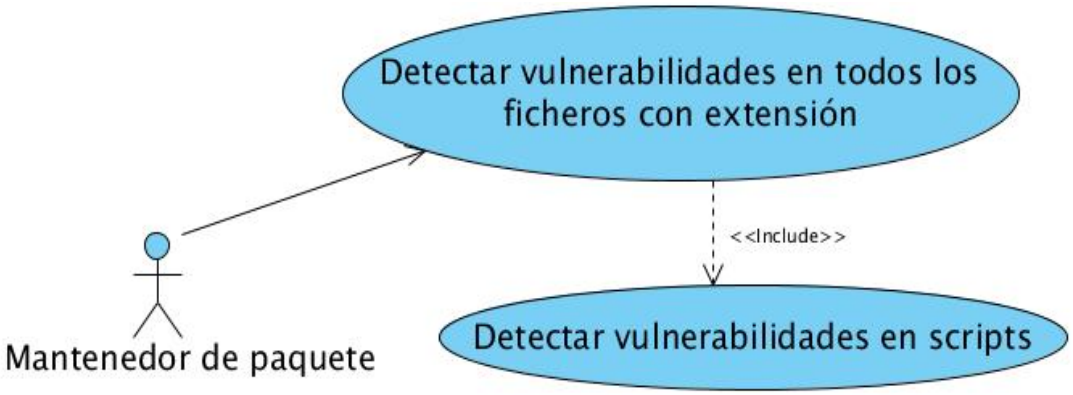

Figura 1: Diagrama de casos de uso.

2.4.3 Descripción textual de los Casos de uso del sistema

A continuación se expone la descripción del caso de uso “Detectar vulnerabilidades de todos los ficheros de código fuente con extensión”.

Nombre de caso de uso: Detectar vulnerabilidades de todos los ficheros de código fuente con extensión.	
Actor(es):	Mantenedor del repositorio.
Propósito:	El caso de uso permite analizar el fichero de código fuente que tenga extensión .py, .c, .sh o .pl.
Resumen:	El caso de uso se inicia cuando el actor presiona el botón “Auditar”, se listarán debajo los ficheros que contienen vulnerabilidades en el código fuente.
Referencia:	RF1.
Prioridad:	Alta.
Curso normal de eventos	
Acción del actor	**Respuesta del sistema**

1. Presiona el botón “Auditar”.	
	2. Valida el campo dirección del repositorio: • Si está vacío ir al flujo alterno 1. • Si no existe o es diferente de la ruta de un archivo o de un directorio ir al flujo alterno 2. • Si existe y es la ruta de un archivo o de un directorio seguir el flujo básico de eventos.
	3. Valida el campo dirección del directorio de reporte: • Si está vacío ir al flujo alterno 3. • Si no existe o es diferente de la ruta de un directorio ir al flujo alterno 4. • Si existe y es la ruta de un directorio seguir el flujo básico de eventos.
	4. Valida el campo dirección y directorio de reporte no posean el mismo valor: • Si los campos son iguales ir al flujo 5.
	5. Realiza la auditoría del repositorio seleccionado a partir de las extensiones (.py, .pl, .c o .sh) de los ficheros. Va analizando fichero a fichero.
	6. Una vez terminada la auditoría: • Si no detecta vulnerabilidades ir al flujo alterno 6. • Si detecta vulnerabilidades seguir el flujo básico de eventos.
	7. Lista los ficheros que deben ser analizados porque contienen vulnerabilidades en el código.
	8. Termina el caso de uso.
Flujos Alternos	
1. Campo dirección vacío.	

Actor	Sistema
	1 Muestra el mensaje “El campo dirección está vacío”.
	2 Termina el caso de uso.
2. Dirección errónea.	
Actor	**Sistema**
	1 Muestra el mensaje “El dato dirección es erróneo”.
	2 Termina el caso de uso.
3. Campo directorio de reporte vacío.	
Actor	**Sistema**
	1 Muestra el mensaje “El campo directorio de reporte está vacío”.
	2 Termina el caso de uso.
4. Directorio de reporte erróneo.	
Actor	**Sistema**
	1 Muestra el mensaje “El dato directorio de reportes es erróneo”.
	2 Termina el caso de uso.
5. Campo dirección y directorio de reporte iguales.	
Actor	**Sistema**
	1 Muestra el mensaje “El dato directorio de reportes no debe coincidir con el dato dirección ”.
	2 Termina el caso de uso.
6. No detecta vulnerabilidades.	
Actor	**Sistema**
	1 Muestra el mensaje “No se detectaron vulnerabilidades”.
	2 Termina el caso de uso.
Prototipo de interfaz.	

Figura 2: Auditar fichero de código fuente.

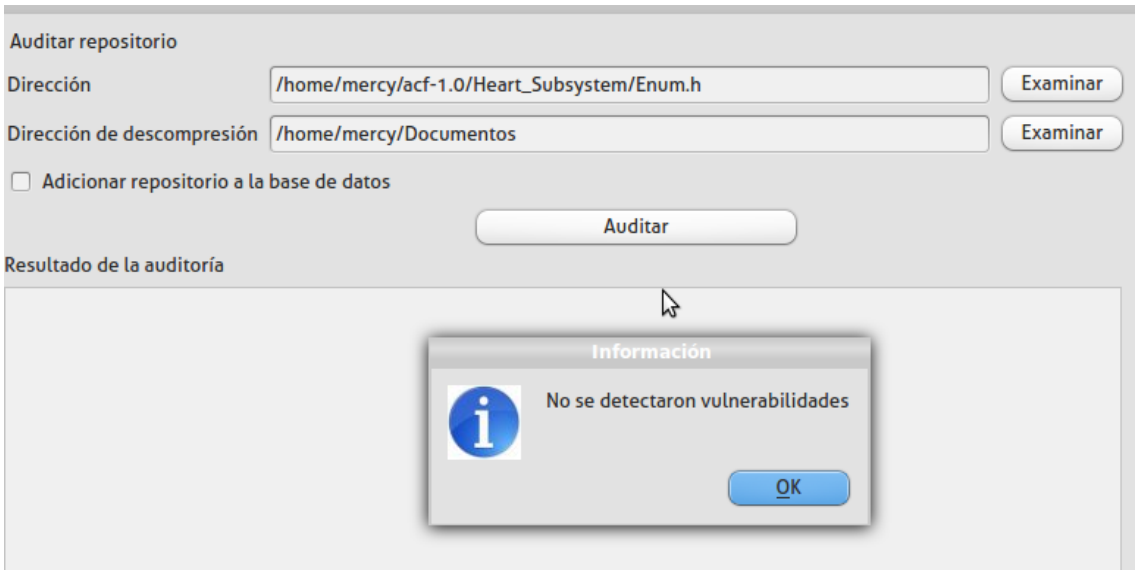

Figura 3: No se detectaron vulnerabilidades en el código fuente.

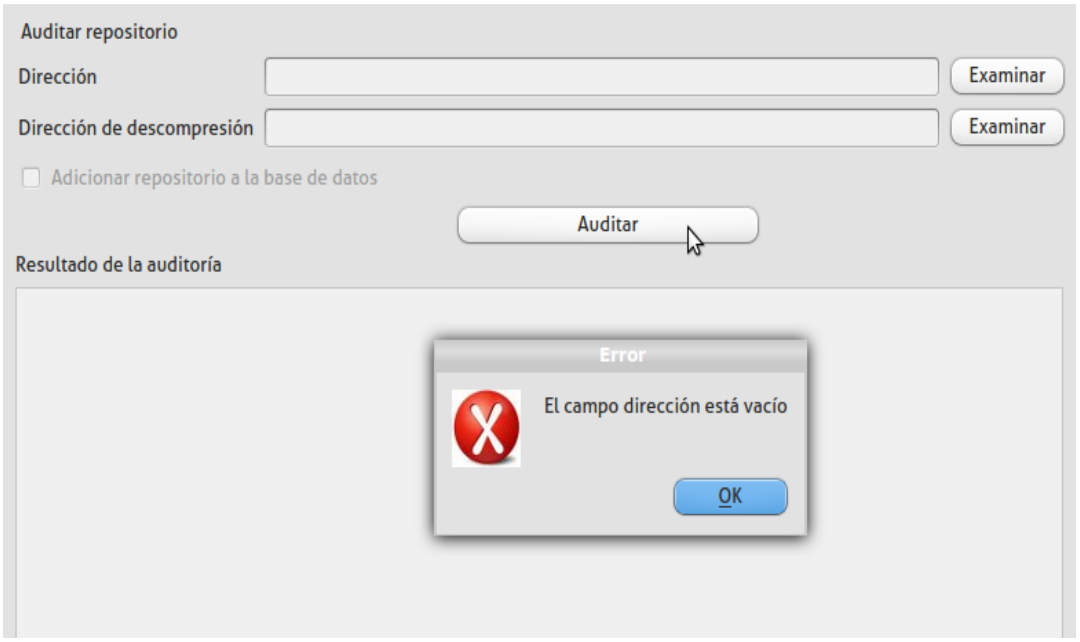

Figura 4: El campo dirección está vacío.

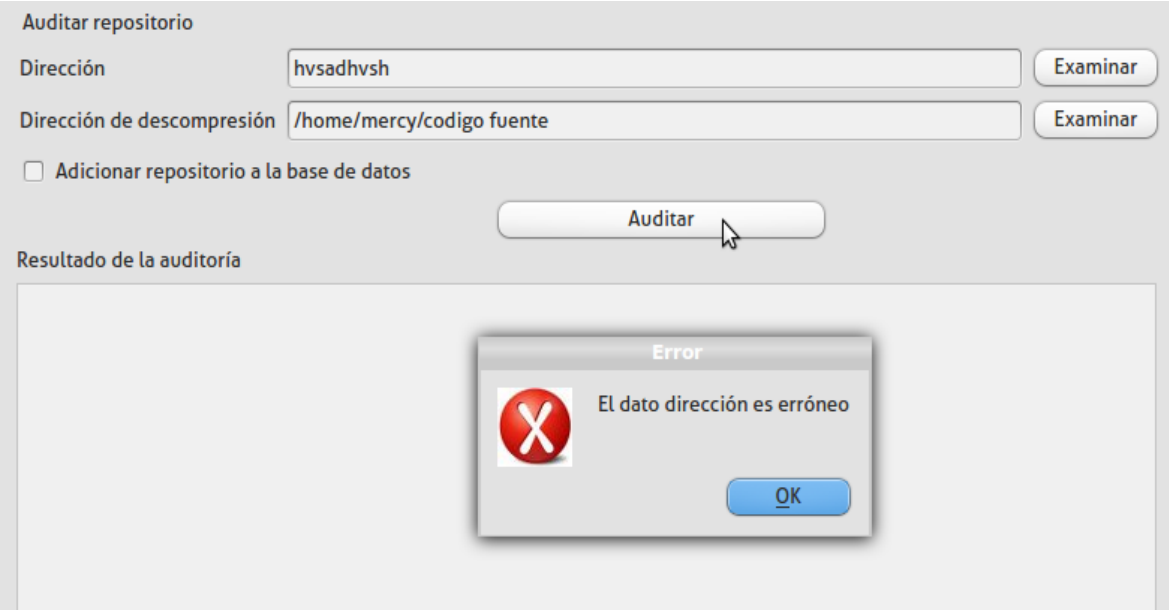

Figura 5: El campo dirección es erróneo.

Tabla 3: Descripción textual caso de uso "Detectar vulnerabilidades de todos los ficheros de código fuente con extensión".

Para ver la descripción de los restantes casos de uso del sistema ir al Anexo 1.

2.5 Diagrama de clases

El diagrama de clases le permite al desarrollador describir la estructura del sistema en cuestión a través de clases orientadas a objetos.

Para ver el diagrama de clases correspondiente a la lógica del negocio de la herramienta ACF v1.1 ir al Anexo 2.

2.6 Diagrama de interacción del diseño

Un diagrama de interacción (secuencia o colaboración) muestra la forma en que un cliente (actor) y otros objetos (clases) se comunican entre sí, en respuesta a un determinado suceso [32].

2.6.1 Diagrama de secuencia

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Se utilizan para mostrar qué objetos se comunican con qué otros objetos y qué mensajes informan dichas comunicaciones, además no están pensados para mostrar lógicas de procedimientos complejos [33].

Para ver el diagrama de secuencia secuencia del CU “Detectar vulnerabilidades de todos los ficheros de código fuente con extensión” ir al Anexo 3.

2.7 Definición de la arquitectura y patrones

En el presente apartado se expondrán los aspectos referentes a la arquitectura con la que cuenta la herramienta en cuestión.

2.7.1 Arquitectura

Roger S. Pressman plantea que la arquitectura de software de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen los componentes del software, las propiedades visibles externamente de los componentes y las relaciones entre ellos [34].

Arquitectura de ACF

La metodología escogida en el presente trabajo (*OpenUp*), recomienda las siguientes vistas para la representación de la arquitectura:

Lógica: Describe la estructura y el comportamiento de porciones arquitectónicamente significativas del sistema. Esto podría incluir la estructura de paquetes, interfaces críticas, clases y subsistemas importantes, y las relaciones entre estos elementos. Incluye vistas físicas y lógicas de datos persistentes, si la persistencia se construirá en el sistema. Este es un subconjunto del diseño documentado [35].

La herramienta ACF v1.1 posee una arquitectura 3 capas (presentación, negocio, acceso a datos). La **capa de presentación** o interfaz gráfica es la encargada de mostrar información al usuario y comunicar la información del usuario en un tiempo mínimo (realizando un filtrado previo para comprobar que no hay errores de formato). La interfaz de la herramienta radica en la carpeta con nombre *Facade_Subsystem,* la cual cuenta con las clases referente a cada una de las interfaces con que cuenta la misma. La **capa de**

negocio contiene toda la lógica, las funcionalidades implementadas, es aquí donde se establecen las reglas a cumplir. En la herramienta en cuestión toda la auditoría transcurre en esta capa, desde el análisis del repositorio hasta la generación de los reportes con los detalles de la misma. La **capa acceso a datos** contiene la base de datos con toda la información que se almacena en el proceso auditor. La comunicación con la base de datos se expresa cuando el usuario solicita guardar la dirección del repositorio que se va a auditar y luego cuando accede a la información (listar, editar, eliminar).

La herramienta ACF v1.0 tiene como característica que la capa de procesamiento cuenta también con una arquitectura basada en componentes. En la herramienta ACF cada uno de los *plugins* de auditoría y descompresión constituyen componentes que se utilizan para el manejo de la información, así como las clases referentes al propio negocio. Ello permite que los componentes se puedan probar por separado, antes de la integración general, así como es fácil de actualizar y/o agregar componentes a medida de la necesidad, sin afectar otras partes del sistema.

Caso de uso: Una lista o diagrama de los casos de uso que contienen requerimientos arquitectónicamente significativos [35].

Para entender esta vista ir al diagrama de CU ya expuesto.

2.7.2 Patrón de diseño

Muchos de los escritores que emprenden el tema de los patrones hacen referencia a la obra del arquitecto Christopher Alexander, quien presentó por primera vez un lenguaje de patrones para poder diseñar ciudades, barrios, grupos de edificios, plazas, edificios, habitaciones.

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, además describe el núcleo de la solución de ese problema de forma que se puede utilizar esta solución un millón de veces, sin hacerlo nunca de la misma manera [36].

Entiéndase por patrón a la pareja de problema/solución con un nombre y que es aplicado a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas [37].

Patrones GRASP

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos,

expresados en forma de patrones. GRASP es el acrónimo para *General Responsibility Assignment Software Patterns* (Patrones Generales de Software para Asignar Responsabilidades). Son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [34].

- Creador.
- Controlador.
- Experto.
- Bajo acoplamiento.
- Alta cohesión.

Patrones GOF

Los patrones GOF ((*Gang of Four*) Pandilla de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y comportamiento [37].

- Creacionales: escriben las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
- Estructurales: Describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades.
- Comportamiento: Definen la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

De manera general estos patrones describen las formas comunes en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros, además tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad [38].

Patrones que se encuentran en ACF

Patrones GRASP:

Creador: El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este

patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento [37].

Beneficios:

- Se brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que nos llevaron a elegirla como el parámetro adecuado [37].

En dicha clase se crean los objetos (*pluginmanagement, fileManager*) que representan las entidades (*ComponentManagement, FileManager*), lo que evidencia que la clase *Seeker* es "Creador" de las entidades.

```
Seeker::Seeker() {
    // TODO Auto-generated constructor stub

    Command command;
    userDir = command.simpleOutput("echo ~/ACF/");
    userDir = userDir.substr(0, userDir.length() - 1);

    pluginmanagement = new ComponentManagement();
    fileManager = new FileManager();


    string baseDir = "/tmp/ACF/";
    fileManager->createDir(baseDir);

    this->globalValues = new struct GlobalValues();
}
```

Figura 6: Fragmento de código de la clase Seeker de la herramienta ACF v1.1.

Controlador: El patrón controlador sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control [37].

Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

Todas las peticiones son manipuladas por un solo controlador (*Validation*). Esta clase es quien se encarga de validar todos los datos, e informar al usuario cuando la información que introdujo no se corresponde con lo que debe especificar.

Bajo acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras, las conoce y recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Este patrón estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento [37].

Beneficios:

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar [37].

En la herramienta el bajo acoplamiento se evidencia en las clases *ComponentManagement*, *Seeker*. Cuando la clase *Seeker* determina que un archivo es de un lenguaje en particular le pide a la clase *ComponentManagement* que invoque al componente que puede procesar dicho lenguaje relacionándose lo mínimo entre sí y de esta manera delega funciones en *ComponentManagement* propiciando el patrón en cuestión.

Alta cohesión: En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuán relacionadas y están enfocadas las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme [37].

Beneficios:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- Soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico [37].

La clase *FileManager* posee funcionalidades que están estrechamente relacionadas, debido a que se trata del trabajo de los directorios y ficheros con los que se va a trabajar en el proceso de la herramienta ACF v1.1.

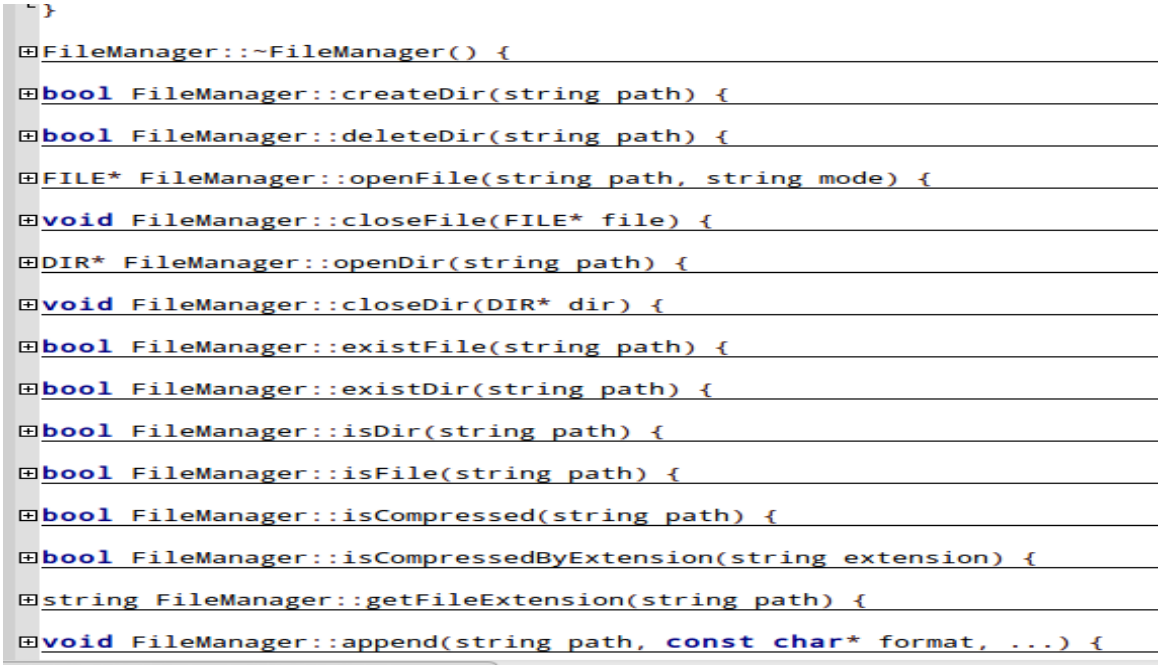

Figura 7: Fragmento de código de la clase FileManager de la herramienta ACF v1.1.

Patrón GOF:

Facade o fachada: Se le da nombre Fachada a la clase definida que ofrece una interfaz común con un conjunto heterogéneo de interfaces. Las interfaces heterogéneas pueden ser un conjunto de funciones, un esquema, un grupo de otras clases o un subsistema [32].

La clase referida define un componente de negocio de alto nivel que contiene y centraliza las interacciones complejas entre los componentes de negocio de nivel inferior. Proporciona a los clientes una única interfaz para la funcionalidad de una aplicación o un subconjunto de la aplicación. Separa los componentes de nivel inferior de negocio, uno del otro, proporcionando un diseño más flexible y comprensible [34].

La clase *Parker* define el patrón fachada ya que es la clase que sirve de interfaz hacia todas las funciones que se pueden ejecutar en la capa lógica del negocio, también la clase *Seecker* constituye la interfaz al

subsistema de auditoría y la clase *DBManagement* constituye la interfaz al subsistema de gestión de la base de datos.

La clase *Parker* es la encargada de conectar la base de datos de la herramienta con el proceso de auditoría. Esto lo hace a través de los métodos que procesan el trabajo de gestión de los datos almacenados del repositorio a auditar (eliminar, editar, mostrar) y de los que procesan el trabajo referente al envío de la dirección donde se encuentra el repositorio que se desea auditar.

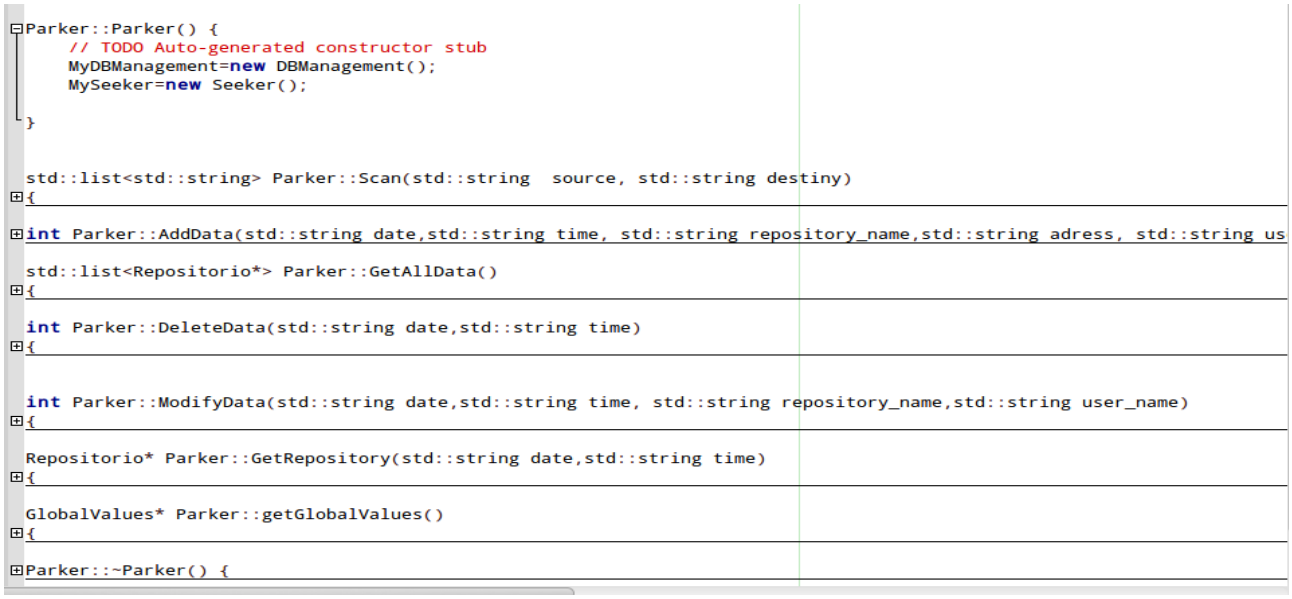

Figura 8: Fragmento de código de la clase Parker de la herramienta ACF v1.1.

2.8 Diagrama de componente

El diagrama de componentes muestra como el sistema está dividido en componentes y las dependencias entre ellos. Un componente representa un elemento físico que forma parte del sistema, se puede representar por nodos y sus operaciones solo se pueden alcanzar a través de interfaces. Este diagrama provee una vista arquitectónica de alto nivel del sistema, y ayuda a los desarrolladores a visualizar el camino de la implementación, permitiendo tomar decisiones respecto a las tareas de implementación [39]. Dicho diagrama también permite modelar las relaciones lógicas entre los componentes del software tales como código fuente, tablas de la base de datos, librerías y paquetes.

Para ver el diagrama de componentes de la herramienta ACF v1.1 ir al Anexo 4.

2.9 Estándares de codificación

En la actualidad es muy común la utilización de estándares de codificación para la mayoría de los lenguajes de programación existentes. La utilización de los mismos permite una mejor comunicación entre los programadores creando condiciones para la reusabilidad y mantenimiento de los sistemas.

Los estándares de código son modelos de programación que no están encaminados a la lógica del programa, sino a su organización y aspecto físico con el objetivo de proporcionar la lectura, comprensión y mantenimiento del código durante el proceso de desarrollo del software. Estos estándares de código establecen las pautas que conlleven a lograr un código más legible y reutilizable, de tal forma que pueda aumentar su mantenibilidad a lo largo del tiempo.

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas.

Para el desarrollo de la herramienta ACF v1.1 se utilizó el estándar para codificación en lenguaje C++.

Estándares empleados:

1. Cada función debe tener un encabezado que contenga:
   - Objetivo de la función y no descripción del procedimiento.
   - Comentarios de apoyo a variables, llamadas a función o inclusión de archivos que no sean obvios al proceso.
   - Explicación de uso de argumentos (parámetros) no obvios.
   - Explicación de uso de valores devueltos (de retorno).
2. Se considera como identificador a los nombres de variables (arreglos, matrices, apuntadores), funciones, así como cualquier tipo de dato definido por el usuario (estructura, clase). Dichos

identificadores deberán seguir las siguientes normas, además de las definidas por el propio lenguaje.

- Deberán tener un nombre significativo para que por su simple lectura, pueda conocerse su función, sin tener que consultar manuales o hacer demasiados comentarios.
- Cada identificador de función, variable o procedimiento deberá ser precedido por la abreviación del tipo de dato de que es la variable, o si se trata de una función o procedimiento del tipo de dato que regresa.

3. Generales:

- No manejar en los programas más de una instrucción por línea.
- Declarar las variables en líneas separadas.
- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.
- Los operadores unarios (++, --, etc.) deben ponerse junto a sus operandos, sin espacios intermedios.
- Antes y después de cada estructura de control se deberá poner una línea en blanco.

4. Paréntesis:

- Para hacer más clara una expresión, es aceptable agregarle paréntesis innecesarios. Dichos paréntesis se llaman paréntesis redundantes [40].

2.10 Cambios realizados a la herramienta ACF v1.0

A la herramienta ACF v1.0 se le hicieron modificaciones para que auditara grandes volúmenes de código fuente y de esta manera auditar todo el repositorio de la distribución de GNU/Linux Nova.

- Se eliminó la inicialización del atributo fileManager en los componentes de auditoría dado que en el constructor de la clase padre *Plugin* se inicializaba y luego en las clases hijas Perl, Python, C, Bash y Cuba se volvía a inicializar.

Las imágenes siguientes muestran el constructor de la clase padre y el constructor de las clases hijas antes del cambio.

```
#include "Plugin.h"

Plugin::Plugin(TipoPlugin t) {

    this->type = t;
    this->fileManager = new FileManager();

}
```

Figura 9: Constructor de la clase padre Plugin.

```
#include "Perl.h"

Perl::Perl(TipoPlugin type, string summarizeDir, string param) :
        Plugin(type, summarizeDir) {

    this->param = param;
    this->tokenizer = new LexicalAnalyser(param);
    this->fileManager = new FileManager();
```

Figura 10: Constructor de la clase hija Perl.

A continuación se muestran fragmentos de los constructores de las clases luego del cambio.

```
#include "Perl.h"

Perl::Perl(TipoPlugin type, string summarizeDir, string param) :
        Plugin(type, summarizeDir) {

    this->param = param;
    this->tokenizer = new LexicalAnalyser(param);
    //this->fileManager = new FileManager();

    this->vulnerable_funct.insert("rmtree");
    this->vulnerable_funct.insert("socket");
    this->vulnerable_funct.insert("connect");
    this->vulnerable_funct.insert("system");
    this->vulnerable_funct.insert("exec");

}
```

Figura 11: Constructor de la clase Perl luego del cambio.

Finalmente este cambio permitió eliminar el consumo de memoria en el sistema al eliminar la asignación

doble de memoria para un mismo atributo.

- Se eliminaron punteros que se quedaban abiertos mientras su función ya había culminado, tomándose como opción de precaución igualarlo al vacío.

Las clases que son evidencia de ello son: Command (stream), ComponentManagement (dirl, fileManager, dire_descompresor, libHandler).

```
try {
    if (libHandler == NULL) {


        libHandler=NULL;
        delete libHandler;
        delete[] err;
        throw string(
                ">>> Imposible cargar la blibloteca : Unable to load library");
```

Figura 12: Cambios realizados en la clase ComponentManagement.

- Se eliminaron arreglos que ya una vez culminada su función no se habían eliminado.

Es evidencia de ello el plugin Cuba (buffer, buffer2), la clase ComponentManagement (err), Seeker (buffer).

- Se añadió una función regfree() que permite liberar la memoria consumida por la función regexec().

```
    res = regexec(&retmp, a.c_str(), (size_t) 1, &mtmp, 0);

    regfree(&retmp);

    if (res==0)
```

Figura 13: Función añadida en el plugin de auditoría Cuba.

- En la clase Cuba se eliminaron 2 llamadas a la función *init_IPTable*(), que se hacían innecesariamente.
- Eliminar variables que trabajan con la gestión de archivos.

Ello se evidencia en las clases *LexicalAnalizer* (*source_file*).

- Se implementó una función *getScriptExtension*(), la cual permite analizar un fichero sin extensión,

pero que posee código fuente. Estos ficheros son llamados *scripts*.

```
string FileManager::getScriptExtension(string path){
// esta función permite leer la cabecera de un script para determinar la extensión del mismo
    string ext = "";
    FILE* p= openFile(path,"r");
    char cabecera[20];
    fgets(cabecera,20,p);
    int pos=-1;
    int longi=0;
    bool valid=cabecera[0]=='#'&& cabecera[1]=='!';
    if(valid){

        for(int i=19;i>=0;i--){
            if(cabecera[i]=='/'){
                pos=i;
                break;
            }
        }
        pos++;

        for(int i=pos ; i<20; i++) {
            if(cabecera[i]==' ')
                break;
            else
                ext+=cabecera[i];
        }

    }

    closeFile(p);
    p=NULL;
    delete p;

    return ext;


}
```

Figura 14: Función implementada en la clase FileManager.

- En la clase Enum se añadió nuevas variables para el manejo de las vulnerabilidades de los lenguajes Php, Java y JavaScript.
- En la clase Seeker se añadieron cambios para que una vez que comience la búsqueda no analice la carpeta debian que contiene los ficheros changelog, rules, copyright y control.

```
void Seeker::SeekDirectory(string &source, string &destiny) {


    string tmpsource="";

    int i=source.length()-1;
    while(source[i]!='/')
    {
        tmpsource+=source[i];
        i--;
    }

    string control, changelog, rules, copyright;
    control = source+"/control";
    changelog = source+"/changelog";
    rules = source+"/rules";
    copyright = source+"/copyright";

    if(tmpsource=="naibed" && fileManager->existFile(control) && fileManager->existFile(changelog) && fileManager->ex
    {
        std::cout<<"salto de la carpeta debian" <<std::endl;
    }
```

Figura 15: Cambio realizado en la clase Seecker.

Conclusiones del capítulo

El desarrollo de este capítulo permitió concluir que:

- ✔ Para lograr un mejor entendimiento del funcionamiento de la herramienta ACF v1.1 y desarrollar la propuesta de solución se identificaron 2 requisitos funcionales y 7 requisitos no funcionales que permitieron el desarrollo de la solución, a fin de cumplir con las necesidades del cliente, se modelaron los escenarios de casos de uso, se definió la arquitectura a usar y se diseñaron los diagramas.
- ✔ Se incorporaron funciones que permitirán a ACF v1.1 auditar *scripts* de los lenguajes Bash, Perl y Python.
- ✔ Se redujo el consumo excesivo de memoria del sistema mediante la liberación de la memoria en tiempo de ejecución.

Capítulo 3: Pruebas realizadas a la herramienta ACF v1.1

Introducción

En el presente capítulo se muestran los diferentes tipos de pruebas, las cuales tributan al correcto funcionamiento de la propuesta de software presentada en esta investigación, y así lograr que se cumpla con las especificaciones propuestas por el cliente.

3.1 Pruebas de software realizadas

El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos pueden estar especificados de forma errónea o imperfecta, así como en posteriores pasos de diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad [34].

Para la evaluación de la calidad de la herramienta ACF v1.1 se realizan las pruebas de caja negra para ejercitar los requisitos funcionales desde el exterior de la misma.

3.1.1 Técnica de pruebas de caja negra: partición equivalente

En el diseño de casos de prueba se utilizó la técnica de partición de equivalencia, que es una técnica del método de prueba de caja negra que divide el campo de entrada de un programa en variables con juegos de datos de entrada y salida.

Roger S. Pressman plantea del método lo siguiente: el método divide el campo de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de pruebas. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de pruebas que hay que desarrollar [34].

En esencia, esta técnica intenta dividir el dominio de entrada de un programa en un número finito de variables de equivalencia, de tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada variable es equivalente a una prueba realizada con cualquier otro valor de dicha variable.

Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Éstas se identifican examinando cada condición de entrada (normalmente una frase en la especificación) y dividiéndola en dos o más grupos. Se definen dos tipos de variables de equivalencia, las válidas, que representan entradas válidas al programa, y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor [41].

3.1.2 Niveles de pruebas

Las pruebas son aplicadas para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Luego de haber concluido la herramienta ACF v1.1, esta fue sometida a pruebas de unidad y pruebas de validación.

- **Pruebas de unidad**

Pressman plantea que en las pruebas de unidad se prueba la interfaz del módulo para asegurar que la información fluya de forma adecuada hacia y desde la unidad de programa que está siendo probada. Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento. Se ejercitan todos los caminos independientes (camino básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y, finalmente, se prueban todos los caminos de manejo de errores [34].

- **Pruebas de sistema en tiempo real**

El software es ensamblado totalmente con cualquier componente de hardware que requiera, se prueba para comprobar que se cumplen los requisitos funcionales. Cualquier elemento del software, desarrollado o adquirido, puede verse como un sistema que debe probarse, ya sea para decidir acerca de su aceptación, para analizar defectos globales o para estudiar aspectos específicos de su comportamiento, tales como seguridad o rendimiento. Este tipo de pruebas estudia el producto completo.

Pruebas de tarea: El primer paso de la prueba de sistemas de tiempo real consiste en probar cada tarea independientemente. Es decir, se diseñan pruebas de caja negra y se ejecutan para cada tarea. Durante

estas pruebas, cada tarea se ejecuta independientemente. La prueba de la tarea ha de descubrir errores en la lógica y en el funcionamiento, pero no los errores de temporización o de comportamiento [34].

Pruebas de comportamiento: Utilizando modelos del sistema creados con herramientas CASE, es posible simular el comportamiento del sistema en tiempo real y examinar su comportamiento como consecuencia de sucesos externos. Estas actividades de análisis pueden servir como base del diseño de casos de prueba que se llevan a cabo cuando se ha construido el software de tiempo real [34].

Pruebas de intertareas: Una vez que se han aislado los errores en las tareas individuales y en el comportamiento del sistema, la prueba se dirige hacia los errores relativos al tiempo**.** Se prueban las tareas asíncronas que se sabe que comunican con otras, con diferentes tasas de datos y cargas de proceso para determinar si se producen errores de sincronismo entre las tareas. Además, se prueban las tareas que se comunican mediante colas de mensajes o almacenes de datos, para detectar errores en el tamaño de esas áreas de almacenamiento de datos [34].

- **Pruebas de validación**

Las pruebas de validación en la ingeniería de software son el proceso de revisión que verifica que el sistema de software producido cumple con las especificaciones y que logra su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones y tutoriales. La validación es el proceso de comprobar que lo que se ha especificado es lo que el usuario realmente quería.

Pruebas Alfa

Esta prueba se lleva a cabo por un cliente en el lugar de desarrollo. Se utiliza el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado [34].

Pruebas Beta

Esta prueba se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente

registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador [34].

3.1.3 Método de prueba de la herramienta: caja negra

La prueba de caja negra intenta encontrar funciones incorrectas o ausentes, errores de interfaz, en estructuras de datos, errores de rendimiento, errores de inicialización y de terminación [34].

Diseño de casos de prueba de caja negra

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados separados, desarrollado para conseguir un objetivo particular o condición de prueba como, por ejemplo, verificar el cumplimiento de un requisito específico [42].

Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

A continuación se representa el caso de prueba que da inicio al CU “Detectar vulnerabilidades de todos los ficheros de código fuente con extensión”.

Descripción de las variables para este caso de prueba:

Variable 1: Dirección donde se encuentra el repositorio o archivo de código fuente que se desea auditar.

Variable 2: Dirección donde se guardarán los reportes de la auditoría.

Escenario	**Descripción**	**Variable 1**	**Variable2**	**Respuesta del sistema**	**Resultado de la prueba**
Escenario 1 [Detectar vulnerabilidades	1.1 El Actor oprime el botón “Auditar”.	V / home/nova/source	V / home/nova/destiny	1.2 El sistema muestra al usuario un resumen de la auditoría en la interfaz. También generará en la dirección contenida en la variable 2 un reporte con los	1-El actor introduce en el campo “Dirección” la dirección donde se encuentran los archivos de código fuente. 2 -El actor introduce en el

Escenario	Descripción	Variable 1	Variable2	Respuesta del sistema	Resultado de la prueba
correctamente].				detalles de la auditoría y otro con un resumen de dicha auditoría.	campo “Dirección de reporte” la dirección del directorio donde se van a almacenar los reportes. 3- Oprimir el botón “Auditar”. 4-El sistema obtiene los datos de los campos de textos. 5-El sistema valida los datos (datos correctos). 5.1- En caso de que exista algún dato incorrecto muestra un mensaje informándolo. 6- El sistema realiza el proceso de auditoría. 7-El sistema muestra en el campo de texto que se encuentra en el centro de la interfaz del sistema una lista con la dirección de los archivos de código fuente que contienen errores, además del número de vulnerabilidades y de
Escenario 2 [Se encuentra un campo vacío].	2.1 El Actor oprime el botón “Auditar”.	“ ”	V / home/nova/destiny	2.2 El sistema emite un mensaje: “El campo dirección está vacío” o “El campo dirección de reporte está vacío”.	
Escenario 3 [La dirección no es válida].	3.1 El Actor oprime el botón “Auditar”.	Cumbres Borrascosas.	V / home/nova/destiny	3.2 El sistema emite un mensaje: “El dato dirección es erróneo” o “El dato dirección de reporte es erróneo”.	
Escenario 4 [La dirección del reporte es la misma que la dirección del directori	4.1 El Actor oprime el botón “Auditar”.	/ home/nova/destiny	/ home/nova/destiny	4.2 El sistema emite un mensaje: “El dato directorio de reporte no debe coincidir con el dato de dirección”.	

Escenario	Descripción	Variable 1	Variable2	Respuesta del sistema	Resultado de la prueba
o a auditar].					advertencias encontradas para cada archivo. 8-En el directorio de reportes se generan dos ficheros con los nombres “ScanSumarize.txt” y “ScanDetails.txt”. Estos ficheros contienen un resumen y la información detallada de los ficheros revisados.

Tabla 4: Caso de prueba para detectar vulnerabilidades en el código fuente.

Para ver la descripción de los restantes casos de prueba ver el 5.

Con el objetivo de comprobar que la herramienta da solución al problema del trabajo en cuestión y así poder concluir la revisión satisfactoriamente de directorios de código fuente de gran tamaño, se tomó una muestra el directorio openssl_1.0.1 y gnome-control-center, que constituyen paquetes de código fuente de gran tamaño.

A continuación se muestra la tabla con los tiempos de ejecución de la herramienta ACF v1.0 y ACF v1.1.

Directorio	Tamaño del directorio	Tiempo de ejecución de ACF v1.0	Tiempo de ejecución de ACF v1.1
openssl_1.0.1	21.4 MB, 2 592 elementos	No termina la ejecución.	8 minutos.
gnome-control-center	37.7 MB 1 105 elementos	No termina la ejecución.	2 minutos.

ACF v1.0	34.9 MB 131 elementos	No termina la ejecución.	Menos de 1 minuto.

Tabla 5: Comparación de tiempos de ACF v1.0 y ACF v1.1 analizando directorios de código fuente (elaboración propia).

También se analizó el repositorio de la rama principal de Ubuntu 14.04 para gnome 3.12 y Nova 2013.

Repositorio	**Tamaño del repositorio**	**Tiempo de ejecución de ACF v1.0**	**Tiempo de ejecución de ACF v1.1**
Rama principal de Ubuntu 14.04 para gnome 3.12	1.9 GB	No termina la ejecución.	3 horas
Nova 2013	15.4 GB	No termina la ejecución.	24 horas

Tabla 6: Comparación de tiempos de la herramienta ACF v1.0 y ACF v1.1 analizando repositorios (elaboración propia).

3.2 Resultado de las pruebas

En las pruebas de caja negras realizadas se identificaron 4 casos de pruebas con un número inicial de 3 iteraciones. Se detectaron un total de 10 no conformidades repartidas en indicadores como: 1) mensajes del sistema, 2) error de funcionalidad, 3) redacción y 4) ortografía.

En la primera iteración se realizaron 4 casos de pruebas detectándose 4 no conformidades, solamente se le dio solución a 3 no conformidades, quedando para la próxima iteración 1 no conformidad. En la segunda iteración se detectaron 4 no conformidades, para un total de 5 no conformidades en la presente iteración, de las cuales fueron resueltas 4. En la tercera iteración se detectaron 2 no conformidades, para un total de 3 no conformidades, siendo todas resueltas.

El total de no conformidades en las 3 iteraciones fue de 10 y todas tuvieron solución. Luego se hicieron 2 iteraciones adicionales para revisar que no existían no conformidades, en la primera de dichas iteraciones se detectó una no conformidad la cual fue resuelta y en la última iteración no se obtuvieron no conformidades. A continuación se representa gráficamente los resultados obtenidos.

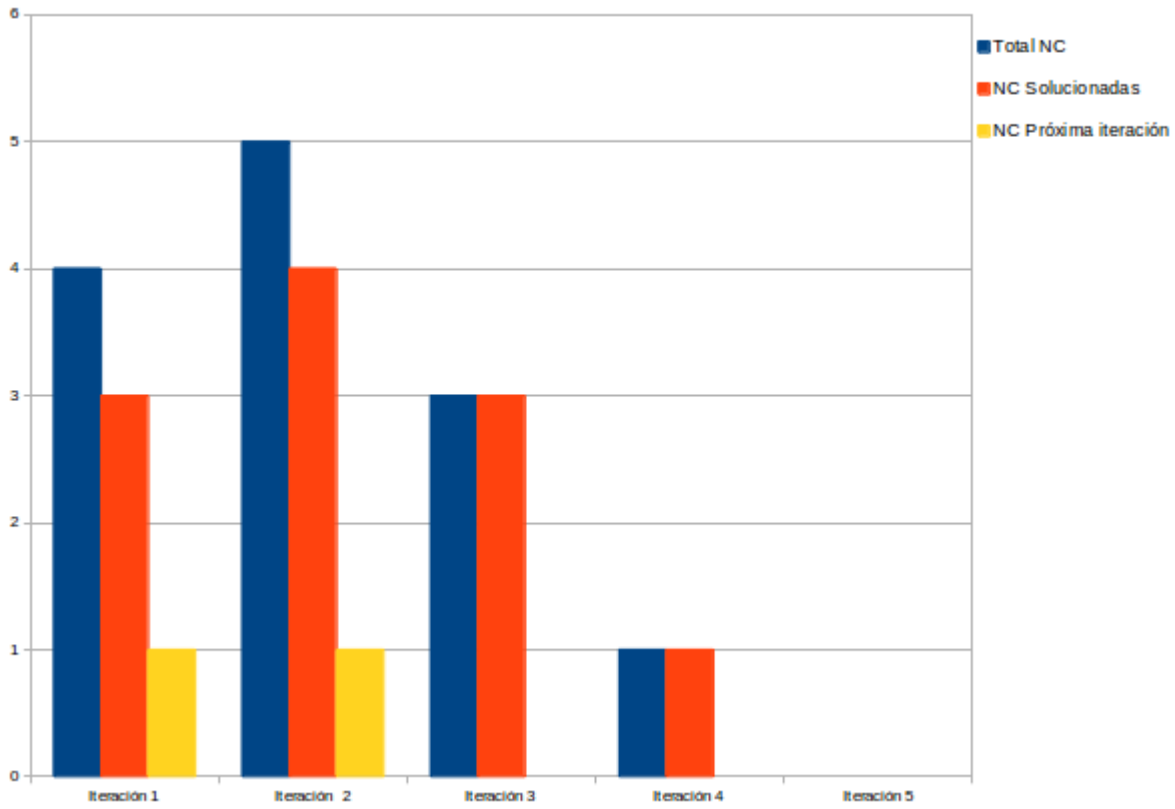

Figura 16: Resultados de la prueba.

3.4 Impacto de la herramienta ACF v1.1

La seguridad de la información es un aspecto que hoy día es importante valorar y tener en cuenta debido al desarrollo de las tecnologías informáticas, ya que esto pone en riesgo la confidencialidad, disponibilidad e integridad del activo más importante de una entidad. Es por ello, que un producto desarrollado en el país ha de proporcionar al usuario confianza.

La ventaja fundamental que provee la herramienta ACF v1.1 es que permite auditar todo el repositorio de la distribución GNU/Linux Nova. A pesar de ser desarrollada para Nova 2013, ACF v1.1, audita la rama principal de Ubuntu 14.04 para Gnome 3.12 que es la base de Nova 2015.

3.5 Conclusiones del capítulo

El desarrollo de este capítulo permitió concluir que:

- ✔ La realización de pruebas con el propósito de eliminar errores detectados fue satisfactoria, pues se corrigieron los fallos en cada una de las iteraciones realizadas.
- ✔ Tras el análisis de los resultados de las pruebas se demostró la solidez de la solución.
- ✔ La herramienta ACF v1.1 está lista para auditar el repositorio de la distribución GNU/Linux Nova.

Conclusiones

- El estudio de las herramientas de código fuente permitió concluir que ninguna de las herramientas de auditoría de código fuente detectan posibles ataques de carácter político, independientemente de los lenguajes que debe analizar.
- El estudio de las herramientas de auditoría de código fuente permitió hacer énfasis en las características de ACF v1.0 posibilitando la definición de 2 requisitos necesarios para desarrollar la herramienta ACF v1.1, siendo el requisito crítico “Detectar vulnerabilidades de todos los ficheros de código fuente con extensión”.
- El desarrollo de las funcionalidades implementadas permitió aumentar la zona de búsqueda de la herramienta ACF v1.1.
- Al ser la herramienta desarrollada capaz de auditar paquetes de gran tamaño se considera lista para ser incorporada en el repositorio de la distribución cubana de GNU/Linux Nova.
- La herramienta ACF v1.1 detectó las vulnerabilidades de todo el repositorio de la distribución GNU/Linux en un plazo de 24 horas.

Recomendaciones

- Dar la posibilidad a los usuarios de que la opción de “Graficar” les permita escoger otros tipos de gráficas, no solamente gráfica de barras.
- Actualizar los *plugins* de auditoría con nuevas funciones para detectar vulnerabilidades.

Referencias Bibliográficas

1—Termes, Rafael. Francis Bacon, 1997. [Disponible en: es.wikiquote.org/wiki/Francis_Bacon]

2--Pierra Fuentes, Allan. Nova, distribución cubana de GNU/Linux. Conceptualización y Reestructuración Estratégica de la Distribución Cubana de GNU/Linux “Nova”. [Tesis de Maestría] Universidad de las Ciencias Informáticas. Cuba. 2011.

3—Perez, Asier. ¿Que son los repositorios en Linux? Útiles, rápidos y seguros. [En línea] [Citado el: 14 de nomviebre 2014] [Disponible en: http://ovtoaster.com/repositorios-linux]

4—Diccionario de informática y tecnología. [En línea] [Citado el: 14 de octubre de 2014] [Disponible en: www.alegsa.com.ar/Dic/codigo%20fuente.php#sthash.Xi3zDaBX.dpuf]

5--Diccionario de informática y tecnología. [En línea] [Citado el: 6 de febrero de 2014] [Disponible en: www.alegsa.com.ar/Dic/bytecode]

6--Diccionario de informática y tecnología. [En línea] [Citado el: 14 de nomviebre de 2014] [Disponible en:http://www.alegsa.com.ar/Dic/codigo]

7--Buguroo bugScout. Analizador de código estático. [En línea] [Citado el: 1 de diciembre de 2014] [Disponible en: www.taboada.info/portfolio/buguroo-bugscout-analizador-de-codigo/]

8—Suarez, Gladys. Buguroo bugScout, 2014. [Disponible en: gpd.sip.ucm.es/sonia/docencia/master1112]

9—Nieto, Javier. Tiger: herramienta de auditoría Unix. [En línea] [Citado el: 8 de agosto de 2013] [Disponible en: https://netjnl.wordpress.com/2013/07/14/tiger-herramienta-de-auditoria-unix/]

10—Tiger, comprueba tu seguridad local. [En línea] [Citado el: 11 de noviembre, 2012] [Disponible en: http://www.nexolinux.com/tiger-comprueba-tu-seguridad-local/]

11—Lynis, auditoría hardening, seguridad de sistemas. [En línea] [Citado el: 8 de julio de 2012] [Disponible en: http://rm-rf.es/lynis-auditoria-hardening-seguridad-sistemas/]

12—Hammurapi, herramienta libre de inspección de código fuente. [En línea] [Citado el: 11 de marzo de 2011] [Disponible en: http://www.javahispano.org/antiguo_javahispano_org/2004/5/17/hammurapi-herramienta-libre-de-inspeccion-de-codigo-fuente-j.html]

13—Herramienta Sonar Qube. [En línea] [Citado el: 1 de junio 2012] [Disponible en: docs.codehaus.org/display/SONAR/User+Guide]

14--Herramienta LClint. [En línea] [Citado el: 23 de junio 2013] [Disponible

en:https://msdn.microsoft.com/es-es/library/bb972268.aspx
15--"Ejemplo de auditoría automatizada: RATS" [En línea] [Citado el: 1 de febrero de 2012][Disponible en: http://www.debian.org/security/audit/examples/RATS]
16--Atacando a Linux Herramientas para realizar auditorías. [En línea] [5 de enero de 2012] [Disponible en: http://www.wikilearning.com/tutorial/atacando_linux-herramientas_para_realizar_auditorias/4250-2]
17--BUGBLOG: "Blog de Blugeroo. Detectar vulnerabilidades en C y C++ con Flawfinder" [En línea] [Citado el: 10 enero de 2011] [Disponible en: http://blog.buguroo.com/=http://backtrackworld.wordpress.com/2008/09/12/detectar-vulnerabilidades-en-c-y-c-con-flawfinder/]
18-- Examples Pscan. [En línea] [Citado el: 17 de mayo de 2011] [Disponible en: http://www.debian.org/security/audit/examples/http://www.debian.org/security/audit/examples/pscan]
19--Metodología. [En línea] [Citado el: 17 de Octubre de 2012] [Disponible en: http://todotecnology.blogspot.com]
20—Metodología OpenUp. [En línea] [Citado el: 11 de diciembre de 2014] [Disponible en: http://openupeaojmp.blogspot.com/2013/09/metodologia-open-up.html]
21--Metodologías de desarrollo de software. [En línea] [Citado el: 10 de enero de 2013] [Disponible en: http://http://es.slideshare.net/carmeloh2/metodologa-open-up-39321348]
22-- Openup. [En línea] [Citado el: 30 de noviembre de 2010] [Disponible en: http://epf.eclipse.org/wikis/openup/]
23—Panizzi, Marisa. Participación de los usuarios en el proceso de desarrollo software, 2013.
24—Visual Paradigm essential. [En línea] [Citado el: 3 de noviembre de 2014] [Disponible en: https://www.udemy.com/visual-paradigm-essential/]
25—QtCreatorWhiteppaper_Spanish. [En línea] [Citado el: 2 de febrero 2012] [Disponible en: http://qt-project.org/wiki/ QtCreatorWhiteppaper_Spanish]
26—Albán, Oscar. Introducción al Qt y al Qt Creator, 2013.
27—Dorias Osvaldo. Doxygen, 2013.
28—Ruz, Francisco. Documentación Qt, 2012.
29—Guilleromo, Cristian. Curso de Lenguajes de Programación, 2013.
30—Gallardo, Mauricio. Manual Cpp, 2010.

31—Códigos sencillos hechos en c. [En línea] [Citado el: 2 de junio de 2013] [Disponible en: https://saforas.wordpress.com/2008/01/04/codigos-sencillos-hechos-en c/]
32--Cáceres Telle, Jesús. Diagrama de secuencia. [Citado el: 20 de febrero de 2013] [Disponible en: www.eumed.net/libros/2009/583/Representacion%20del%20modelo%20de%20dominio.htm]
33--Sparxsystems. [En línea] [Citado el: 19 de Febrero de 2013] [Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.html]
34—Pressman, Roger S. INGENIERÍA DEL SOFTWARE UN ENFOQUE PRÁCTICO, Quinta edición.
35—Openup. [En línea] [Citado el: 3 de marzo de 2015] [Disponible en: http://epf.eclipse.org/wikis/openup/index.htm]
36—Christopher, Alexander. Patrones, 1977 .
37—Arquitectura de 3 capas. [En línea] [Citado el: 3 de marzo de 2012] [Disponible en: http://www.buenastareas.com/ensayos/Arquitectura-De-3-Capas/3936241.html]
38--ComponentSource, What are components?. [En línea] [Citado el: 30 de agosto de 2011] [Disponible en: http://www.componentsource.com/Services/WhatAreComponents.asp?bhcp=1]
39—WebCabComponents,AboutComponentBasedDevelopment. [En línea] [Citado el: 2 de mayo de 2012] [Disponible en: http://webcabcomponents.com/componentization.shtml]
40—Departamento de electrónica, sistemas e informática. Coordinación docente de programación estándar para codificación en lenguaje C++. Estándar de codificación de lenguaje C++.
41-- Arquitectura basada en componentes. [En línea] [Citado el: 2 de febrero de 2015] [Disponible en:http://es.slideshare.net/urumisama/arquitectura-basada-en-componentes]
42---Larman, C,UML y Patrones. Introducción al análisis y diseño orientado a objetos, 2011.

Bibliografía

1—Medina, Eduardo. Metodologías de Desarrollo de Software. [En línea] [Citado el: 23 de marzo de 2012] [Disponible en: http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI /Tema04.pdf]

2--Marrero López, Yadira y Rosales García, Adonis R. Propuesta del diseño arquitectónico de la. s.l.: Universidad de las Ciencias Informáticas, 2007.

3—Dosideas. [En línea] [Citado el: 4 de enero de 2012] [Disponible en: http://www.dosideas.com/wiki/IDE]

4--"IEEE Standar Glossary Software Engineering Terminology"[En línea] [Citado el: 2006] [Disponible en: http://standards.ieee.org/reading/ieee/std_public/description/se]

5—López, Mario. Grupo de investigación en reutilización y orientación al objeto. Análisis y Diseño Orientado al Objeto para Reutilización, 2013.

6—Bertrand, Meyer. Construcción de Software Orientado a Objetos. Segunda Edición, 1999.

7--LAMADRID CUESTA, Adrián Serafín. Herramienta para la detección de vulnerabilidades en el código fuente del repositorio de GNU/Linux Nova. [Tesis de Diploma] Universidad de las Ciencias Informáticas. Cuba, 2012.

8--Hernández Sampieri, Roberto, Fernández-Collado, Carlos y Baptista Lucio, Pilar. Metodología de la Investigación. Cuarta Edición. Mexico DF:Mc Graw Hill, 2006. págs. 157-232.

9—Larman, Craig. Uml y patrones, 1999.

10—Ariza, Maribel y Molina Juan Carlos. Introducción y principios básicos del desarrollo basado en componentes, 2011.

11--Ricardo Grau, Cecilia Correa, Mauricio Rojas. Metodología de la Investigación,Segunda edición 2004, Universidad de Ibagué, Coruniversitaria, Editorial Coruniversitaria El POIRA Editores, S. A., Ibagué, Tolima. Colombia.

12--Martínez, J.S. Una Arquitectura para una Herramienta de Diseño. Universidad de Murcia: Departamento de Informática y Sistema.

13--R. Edward Freeman, Alexander Moutchnik. Stakeholder management and CSR: questions and answers. In: UmweltWirtschaftsForum, Springer Verlag, Vol. 21, Nr. 1.

14--"Atacando a Linux Herramientas para realizar auditorías". [En línea] [Citado el: 4 de diciembre de 2012] [Disponible en: http://www.wikilearning.com/tutorial/atacando_linux-

herramientas_para_realizar_auditorias/4250-2]
15--Dwheeleer: FlawFinder [En línea] [Citado el: 11 febrero 2012] [Disponible en: http://www.dwheeler.com/flawfinder/]
16-- Lenguajes-de-programacion. [En línea] [Citado el: 11 marzo 2011] [Disponible en: http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml]
17-- Boletín de vulnerabilidades. [En línea] [Citado el: 18 marzo 2011] [Disponible en: https://www.ccn-cert.cni.es/index.phpoption=com_vulnerabilidades&task=view&id=5873&Itemid=96&lang=es]

Anexos

Anexo 1

Nombre de caso de uso: Detectar vulnerabilidades en *scripts*.	
Actor(es):	No posee actor.
Propósito:	El caso de uso permite analizar *scripts* que contiene código fuente.
Resumen:	El caso de uso se inicia cuando el fichero no posee extensión .py, .c, .sh o .pl.
Referencia:	RF2.
Prioridad:	Alta.
Curso normal de eventos	
Acción del actor	**Respuesta del sistema**
	1. Abre el fichero y lee la cabecera del mismo. **2.** Si esta cabecera comienza con los caracteres #! ir al flujo alterno 1
	3. Una vez terminada la auditoría: • Si no detecta vulnerabilidades ir al flujo alterno 2. • Si detecta vulnerabilidades seguir el flujo básico de eventos.
	4. Lista los ficheros que deben ser analizados porque contienen vulnerabilidades en el código.
	5. Termina el caso de uso.

Flujos Alternos	
1. Cabecera comienza con los caracteres #!.	
Actor	**Sistema**
	1 Lee de atrás hacia delante lo que está al final de la cabecera hasta el carácter /.
	2 Añade esta extensión a la lista para analizar el *script.*
	3 Termina el caso de uso.
2. No detecta vulnerabilidades.	
Actor	**Sistema**
	1 Muestra el mensaje “No se detectaron vulnerabilidades”.
	2 Termina el caso de uso.

Tabla 7: Descripción textual caso de uso “Seleccionar repositorio a auditar”.

Anexo 2

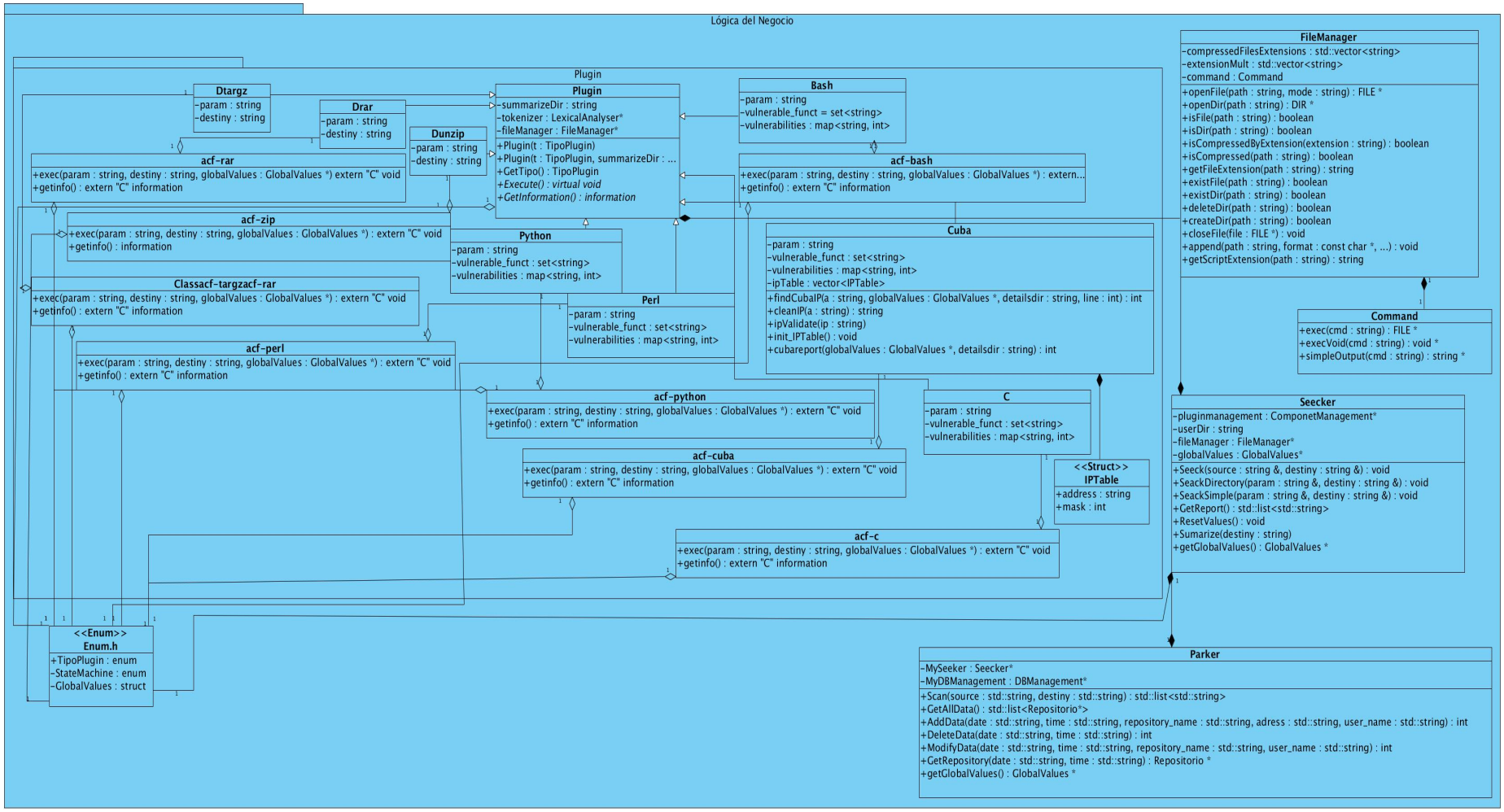

Figura 17: Diagrama de clases.

Anexo 3

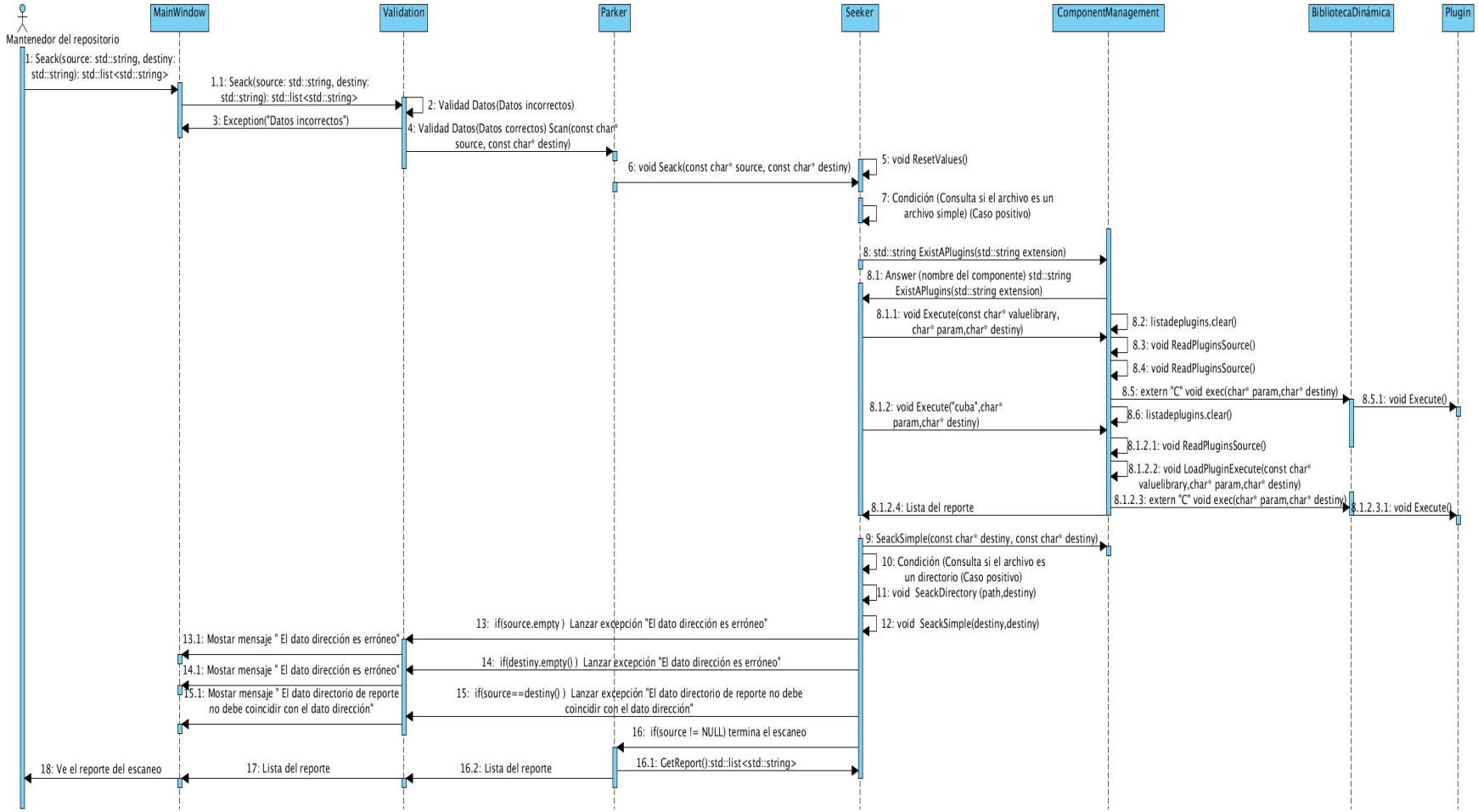

Figura 18: Diagrama de secuencia del CU "Detectar vulnerabilidades de todos los ficheros código fuente con extensión".

Anexo 4

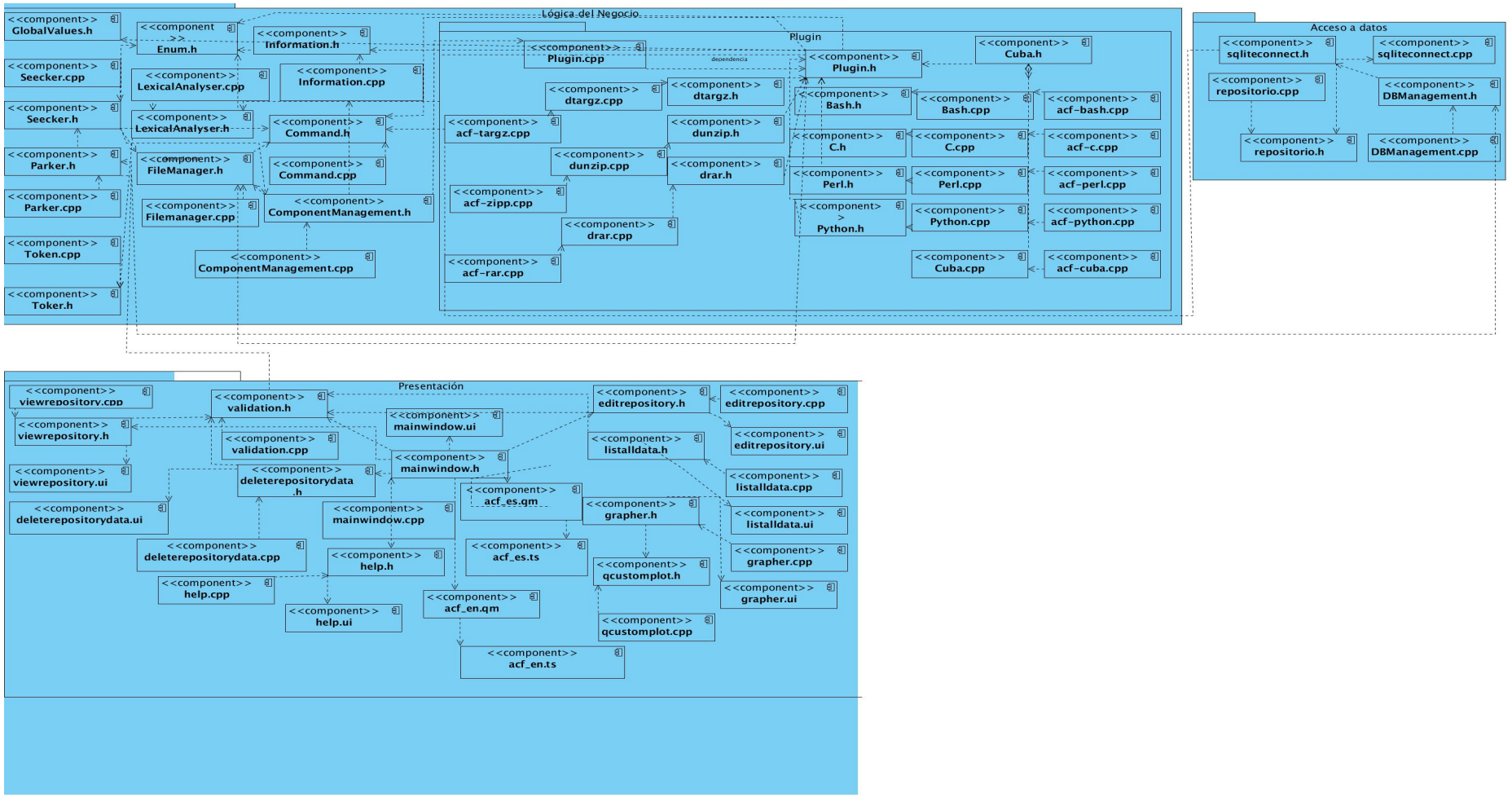

Figura 19: Diagrama de componentes de la herramienta ACF v1.1.

Anexo 5

A continuación se presentan las tablas de casos de pruebas.

Variable 1: Dirección donde se encuentra el repositorio o archivo de código fuente que se desea auditar.

Escenario	**Descripción**	**Variable 1**	**Respuesta del sistema**	**Resultado de la prueba**
Escenario 1 [Especificar ruta del archivo o directorio de código fuente a auditar correctamente].	1.1. El actor introduce en el campo “Dirección” la dirección donde se encuentran los archivos de código fuente que se desean auditar. Para ello puede escribir la dirección manualmente o puede seleccionar el directorio al presionar el botón “Examinar”. También puede el actor	V / home/nova/source	1.2. La herramienta incluye los ficheros seleccionados en la lista de ficheros a revisar, y si se ha seleccionado un directorio, se incluirán en la lista los ficheros contenidos en dicho directorio.	1-El actor introduce en el campo “Dirección” la dirección donde se encuentran los archivos de código fuente.

Escenario	Descripción	Variable 1	Respuesta del sistema	Resultado de la prueba
	seleccionar un único elemento al oprimir el botón “Cancelar”.			

Tabla 8: Caso de prueba “Seleccionar repositorio a auditar”.

Variable 2: Dirección donde se guardarán los reportes de la auditoría.

Escenario	Descripción	Variable 2	Respuesta del sistema	Resultado de la prueba
Escenario 1 [Seleccionar archivos correctamente].	1.1. El actor introduce en el campo “Dirección de reporte” la dirección donde se encuentran los archivos de código fuente que se desean auditar. Para ello puede escribir la	V / home/nova/destiny	1.2. La herramienta actualizará el directorio de salida para los ficheros revisados, que inicialmente es tmp.	1 -El actor introduce en el campo “Dirección de reportes” la dirección del directorio donde se van a almacenar los reportes.

Escenario	Descripción	Variable 2	Respuesta del sistema	Resultado de la prueba
	dirección manualmente o puede seleccionar el directorio al presionar el botón “Examinar”.			

Tabla 9: Caso de prueba “Especificar directorio de reporte”.

Escenario	Descripción	Respuesta del sistema	Resultado de la prueba
Escenario 1 [Seleccionar la opción “Graficar”].	1.1 El actor elige la opción “Ver gráfica de resumen de la revisión” del campo “Graficar”.	1.2. La herramienta muestra una imagen con una gráfica de barras donde se muestra las advertencias y vulnerabilidades de los lenguajes que se encontraron en la revisión realizada con anterioridad.	1-El actor elige la opción “Ver gráfica de resumen de la revisión” del campo “Graficar”. 2- Se muestra una imagen con la grafica de barras que especifica los lenguajes detectados en la revisión así como el número de las advertencias y vulnerabilidades encontradas.

Tabla 10: Caso de prueba “Graficar resultado del repositorio auditado”.

Glosario de términos

Alto nivel: se caracteriza por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de la capacidad ejecutora de las máquinas.

ANSI* (*American National Standards Institute* por sus siglas en inglés)*: es una organización sin fines de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos. ANSI es miembro de la Organización Internacional para la Estandarización (ISO) y de la Comisión Electrotécnica Internacional (*International Electrotechnical Commission*, IEC). La organización también coordina estándares del país estadounidense con estándares internacionales, de tal modo que los productos de dicho país puedan usarse en todo el mundo.

Bajo nivel: es aquel en el que sus instrucciones ejercen un control directo sobre el hardware y están condicionados por la estructura física de la computadora que lo soporta. El uso de la palabra *bajo* en su denominación no implica que el lenguaje sea inferior a un lenguaje de alto nivel, si no que se refiere a la reducida abstracción entre el lenguaje y el hardware.

Bytecode: el código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina [5].

***Enterprise JavaBeans* (EJB):** son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE) de Oracle Corporation. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor, que son, precisamente, los EJB: transacciones, control de la concurrencia, seguridad y otros.

Errores comunes: división por cero, ciclo infinito, problemas aritméticos como desbordamientos (*overflow*) o subdesbordamientos (*underflow*), exceder el tamaño del arreglo, utilizar una variable no inicializada, acceder a memoria no permitida (violación de acceso), pérdida de memoria (*memory leak*), desbordamiento o subdesbordamiento de la pila (estructura de datos), desbordamiento de búffer (*buffer overflow*), bloque mutuo (*deadlock*) .

Falsos positivos: consisten en alarmas que tienen lugar cuando en realidad no se está produciendo ninguna intrusión. Lo más negativo de esta cuestión es que la continua aparición de falsos positivos puede hacer que un administrador acabe ignorando las alarmas, que es igual de negativo que no recibirlas.

GNU (*General Public License* por sus siglas en ingés) o Licencia Pública General de GNU: es la

licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software.

Hacking ético: es una forma de referirse al acto de una persona usar sus conocimientos de informática y seguridad para realizar pruebas en redes y encontrar vulnerabilidades, para luego reportarlas y que se tomen medidas, sin hacer daño. La idea es tener el conocimiento de cuales elementos dentro de una red son vulnerables y corregirlo antes que ocurra hurto de información, por ejemplo.

Internet: conjunto descentralizado de redes de comunicación conectados entre sí. Utilizan la familia de los protocolos TCP/IP, permitiendo que las redes funcionen como un todo y tengan alcance mundial.

Inyección de código SQL: burlar la información a entrar, entrando comandos de código SQL para apoderarse de datos privados de la aplicación (datos no confiables). Ocurre cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta SQL, los datos hostiles del atacante pueden engañar al intérprete en ejecutar comandos no intencionados o acceder datos no autorizados.

ISO (International Organization for Standardization **por sus siglas en inglés*)* Organización Internacional de Normalización:** es el organismo encargado de promover el desarrollo de normas internacionales de fabricación (tanto de productos como de servicios), comercio y comunicación para todas las ramas industriales. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones (públicas o privadas) a nivel internacional.

Proceso de unificado ágil: es la versión simplificada de RUP. La misma describe de manera simple la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

Plugins: Un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.