Universidad de las Ciencias Informáticas
Facultad 1

"***Herramienta para la gestión de perfiles de AppArmor para GNU/Linux Nova servidores.***"

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Asney Hidalgo Palmero

Tutores: Ing. Gustavo Quezada Arévalo

Ing. Yazmin Suárez Suárez

La Habana, 2017

“Necesitamos reforzar el espíritu de colaboración de la gente, respetando su libertad para cooperar y evitando imponer esquemas para dividirlos y dominarlos.”

Richard Stallman.

Declaración de autoría

Declaro por este medio que yo Asney Hidalgo Palmero, con carné de identidad 91022128825 soy el autor principal del trabajo titulado Herramienta para la gestión de perfiles de seguridad de AppArmor para GNU/Linux Nova servidores y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firman la presente a los ____ días del mes de _____ del año _____.

_____

Asney Hidalgo Palmero

Autor

_____ _____

Ing. Yazmin Suárez Suárez

Tutor

Ing. Gustavo Quezada Arévalo

Tutor

Agradecimientos

Me gustaría agradecerle a mi papá y mi mamá que son las personas más especiales de mi corazón.

A mis compañeros de aula que todos estos años me han sabido apoyar en todas mis decisiones.

A mis mejores amigos del apartamento y en general para los del edificio.

A todos los profesores que en el transcurso de la carrera han sabido formarme como profesional.

A mis tutores que han estado a mi lado, durante todo el proceso de mi 5to año ayudándome a cumplir mi sueño.

A mi oponente que me ha brindado la mano para ayudarme a salir adelante con los tropiezos que he encontrado.

A mi tribunal en todos los cortes y en la defensa por alentarme a superarme cada vez más, en aras de lograr mi meta.

Dedicatoria

El presente trabajo está dedicado a las dos personas a las cuales les debo la vida y gracias a los cuales pude cumplir mi sueño, mi mamá y mi papá. Ambos supieron estar a mi lado en el transcurso de toda la carrera, y más importante aún fueron los que supieron brindarme su mano para levantarme en los tropiezos que tuve mientras recorría la carrera. Fueron ellos siempre mi luz al final del camino, guiándome para demostrarme a mí mismo que si se puede, que todo es cuestión de proponérselo, y como bien me enseño mi papá, y cito: ¨Una persona no puede ser aprendiz de todo y maestro de nada¨.

Resumen

Con el surgimiento de GNU/Linux Nova en Cuba y en aras de lograr la independencia tecnológica, surge la necesidad de garantizar la protección de la información y la seguridad del sistema. GNU/Linux Nova en su variante para servidores cuenta con el *framework* de seguridad AppArmor, encargado de controlar la manera en que se ejecutan las aplicaciones en el sistema operativo mediante perfiles de seguridad. El mismo no cuenta con un mecanismo que se encargue de su configuración, surgiendo la necesidad de crear una herramienta informática que permita la gestión de los perfiles de seguridad del sistema. La misma formará parte de nova-manager, gestor de administración de GNU/Linux Nova en su versión para servidores, garantizando la configuración de las reglas, capacidades y permisos de lectura, escritura, y ejecución sobre los directorios en los perfiles. Para esto, se realizó un estudio de las herramientas de administración de los *framework* de seguridad de las distribuciones de GNU/Linux. El módulo fue creado siguiendo las pautas que rige la metodología AUP en su variación para la UCI y desarrollado en el lenguaje de programación Python utilizando como entorno de desarrollo integrado PyCharm, modelado a través de VisualParadigm utilizando UML como lenguaje de modelado. La validación del módulo se realizó mediante pruebas unitarias, pruebas de integración y aceptación. Con la validación se garantizó el correcto funcionamiento del módulo, resolviendo los problemas existentes y garantizando la creación de nuevos perfiles de seguridad, con el objetivo de proteger al sistema operativo de ataques internos, externos y aplicaciones maliciosas.

Palabras clave: *apparmor, distribuciones, herramienta, GNU/Linux, nova-manager, perfil, sistema operativo*

Índice

Índice de figuras

Índice de tablas

Introducción

Con el comienzo del siglo XXI cobran un mayor auge el uso de las Tecnologías de la Información y las Comunicaciones (TIC) en las empresas y centros de desarrollo, para la producción de software de computadoras y en la evolución de nuevas tendencias a Sistemas Operativos (SO). Como parte de este progreso y para que los desarrolladores y la comunidad de usuarios de estos SO pudiesen contribuir a mejorar las aplicaciones y el mismo SO, surge el *software* libre brindando a la comunidad de usuarios la libertad de ejecutarlos, copiarlos, modificarlos y distribuirlos libremente (1).

Esta posibilidad de poder contribuir a las mejoras del SO, trajo consigo que se esparciera una amplia gama de distribuciones libres. Debido a sus múltiples ventajas y en aras de mejorar la seguridad de las empresas y lograr la independencia tecnológica, Cuba se traza la meta de crear su propia distribución. La distribución cubana GNU/Linux Nova es desarrollada en el Centro de Software Libre (CESOL) perteneciente a la Universidad de las Ciencias Informáticas (UCI) y constituye el pilar fundamental en la migración a tecnologías de código abierto en los Organismos de Administración Central del Estado (OACE). Esta distribución en su variante para servidores cuenta con nova-manager, la cual es una herramienta para administrar los servicios. Posee además dentro de su sistema de seguridad, el *framework* AppArmor, heredado de la distribución *Ubuntu* de *GNU/Linux* (2).

AppArmor se encarga de establecer restricciones de acceso sobre las aplicaciones basándose en políticas del sistema, implementadas como perfiles que a su vez son usados para especificar las acciones que podría o no realizar una aplicación en el sistema. Las restricciones sobre las aplicaciones se basan en las políticas del sistema y no en los permisos del usuario, logrando de esta manera el control del funcionamiento sobre el SO. Estas políticas se encuentran descritas a través de perfiles, por los cuales se rige AppArmor para controlar el desempeño de las aplicaciones. Por otro lado, el cumplimiento de las políticas es responsabilidad del propio *Kernel Linux* y no de una aplicación o servicio, basándose en el control de acceso y permitiendo especificar qué archivo puede ser leído, escrito y ejecutado en cada programa. AppArmor protege las aplicaciones mediante la imposición de buenas prácticas de comportamiento, de forma que se puedan prevenir ataques a nivel de software (3).

Los perfiles de AppArmor se encargan de controlar las acciones que pueden realizar las aplicaciones sobre el sistema. Estos perfiles no contienen todas las restricciones que podrían tener para cada aplicación en el sistema, ni tampoco existen perfiles para todas las aplicaciones. La

creación o modificación de los mismos se realiza de forma manual, causando que cualquier modificación de seguridad que se necesite realizar sobre la aplicación, requerirá de un largo período de tiempo por la complejidad y variedad de configuraciones que se pueden aplicar, como a su vez será necesario que el administrador cuente con un amplio dominio acerca del funcionamiento y la manera en que se encuentran estructurados los perfiles.

Un ejemplo claro de la protección que brindan los perfiles sobre el SO, puede ser descrito de la siguiente manera: Cuando una aplicación del SO requiere de permisos de administración para su ejecución, puede realizar cualquier cambio sobre el mismo, si su código fuente es alterado para realizar acciones indebidas sobre el SO con el fin de eliminar información del sistema de archivos, abrir puertos del cortafuego, agregar usuarios administradores, entre otros. Esto se puede evitar creándole un perfil de seguridad a la aplicación, indicándole que no pueda realizar estas acciones sobre los recursos que se desean proteger, y de esta forma la aplicación no podrá realizar el ataque sobre el sistema.

A partir de los despliegues realizados con Nova servidores en la migración de varios Organismos de la Administración Central del Estado (OACE) los especialistas en migración de servicios telemáticos y los propios administradores de red de las instituciones han mostrado desconocimiento en la correcta aplicación de las políticas de seguridad, los resultados de una entrevista a varios implicados en estos roles ha arrojado que la no aplicación de políticas de control a las aplicaciones del sistema se debe a la complejidad y el alto grado de conocimientos que requiere el trabajo con AppArmor. Lograr mayores niveles de seguridad en Nova servidores requiere que los perfiles de AppArmor se encuentren estructurados correctamente.

Por lo antes descrito se define como **problema científico:** ¿Cómo facilitar la gestión de los perfiles de seguridad del *framework* AppArmor en la distribución cubana GNU/Linux Nova en su variante para servidores?

Se plantea como **objeto de estudio** las herramientas para la gestión de perfiles de seguridad en las distribuciones de GNU/Linux, enmarcado en el **campo de acción** las herramientas para la gestión de perfiles de seguridad en la distribución GNU/Linux Nova en su variante para servidores.

La presente investigación tiene como **objetivo general:** Desarrollar un módulo para la herramienta nova-manager que permita la gestión de los perfiles del *framework* de seguridad AppArmor de la distribución GNU/Linux Nova en su variante para servidores.

Para darle cumplimiento al objetivo general se definen los siguientes **objetivos específicos:**

1. Caracterizar las herramientas de administración de *framework* de seguridad.
2. Diseñar un módulo para la gestión de los perfiles del *framework* de seguridad AppArmor.
3. Implementar el módulo para la gestión de los perfiles del *framework* de seguridad AppArmor.
4. Comprobar el correcto funcionamiento del módulo implementado.

A partir del desarrollo de la investigación surgen una serie de preguntas científicas que ayudan en el proceso de desarrollo:

- ¿Cuáles es la importancia y de qué forma funcionan los *framework* de seguridad en los sistemas operativos GNU/Linux?
- ¿Por qué es necesario incluir en GNU/Linux Nova en su variante para servidores, un módulo que permita la gestión de los perfiles del *framework* de seguridad?
- ¿Cómo incluir el módulo al sistema operativo GNU/Linux Nova en su variante para servidores?
- ¿Qué métodos o técnicas aplicar para la validación del módulo para la gestión de perfiles del *framework* de seguridad AppArmor?

Métodos teóricos

Histórico-lógico: Con la utilización de este método se conocerá la evolución que han tenido los mecanismos de seguridad de las distribuciones *GNU/Linux*, causas de su surgimiento, características que aún mantienen y nuevas evoluciones de las mismas.

El método es aplicado en el diseño teórico del presente trabajo, donde se realiza un estudio del surgimiento de la distribución cubana GNU/Linux Nova, así como de donde proviene el framework presente en la distribución.

Analítico-sintético: Utilizado para dividir el problema en subproblemas, facilitando el estudio de las herramientas de gestión de perfiles de seguridad de las distribuciones de *GNU/Linux* y luego sintetizarlo en una solución general acorde al problema propuesto.

El método se evidencia en la presente investigación, la cual fue dividida en tres capítulos como subproblemas, fundamentación teórica, análisis y diseño e implementación y prueba, para dar solución el problema planteado.

El presente documento está estructurado de la siguiente manera:

Capítulo 1: Abarcará el estudio de las herramientas de administración de perfiles del *framework* de seguridad AppArmor, realizando un análisis de sus distintas características, funciones, ventajas y desventajas. Se realiza además una comparación de estas características con el propósito de desarrollar un módulo capaz de gestionar los perfiles de AppArmor con la mayor fortaleza y seguridad posible.

Capítulo 2: Se presentará una propuesta del módulo que se desea desarrollar, definiendo así, requisitos funcionales y no funcionales, historias de usuarios, la arquitectura que será utilizada, y los patrones de diseño que serán utilizados.

Capítulo 3: Se presentará el estándar de codificación para la implementación del módulo para la gestión de perfiles del *framework* de seguridad. Se definirán las pruebas a realizar, sus métodos, tipos de pruebas y técnicas a emplear. Se mostrará el proceso de aplicación de las pruebas y los resultados que las mismas arrojaron.

Capítulo 1: Fundamentación teórica

En el marco del estudio del estado del arte, se hará referencia a conceptos, características y funcionamiento de las herramientas de gestión de perfiles del *framework* de seguridad AppArmor. A partir de los resultados se analizan diferentes lenguajes y herramientas que sean factibles para desarrollar una solución al problema planteado, así como la metodología para guiar el proceso de desarrollo.

1.1 Conceptos fundamentales

Para una mejor comprensión, en los siguientes epígrafes se definen los conceptos fundamentales de la investigación.

1.1.1 *Framework* (infraestructura, marco)

Un *framework* es una estructura conceptual y tecnológica de soporte, definida normalmente con artefactos o módulos de software concretos en base a la cual otro proyecto de software puede ser organizado y desarrollado. Estos pueden incluir soporte de programas, bibliotecas, lenguajes interpretados, entre otras aplicaciones para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo que extiende o utiliza las aplicaciones del dominio (4).

1.1.2 Seguridad

La Organización Internacional de Normalización (ISO por sus siglas en inglés) hace referencia a que la seguridad consiste en *minimizar la vulnerabilidad de bienes y recursos*. La seguridad en un sistema se basa en los mecanismos de protección que proporciona. Estos mecanismos deben permitir controlar qué usuarios tienen acceso a los recursos del sistema y qué tipo de operaciones pueden realizar sobre esos recursos (5). Para controlar el acceso de los dominios a los recursos se utilizan las Listas de Control de Acceso (*ACL por sus siglas en inglés*) por cada recurso. La *ACL* especifica qué dominios tienen acceso a los recursos y qué operaciones asociadas al recurso pueden utilizar (5).

1.2 Características de los perfiles de AppArmor

1.2.1 Creación de perfiles

Los perfiles de AppArmor son archivos de texto plano alojados en la ruta /etc/apparmor.d/. y contienen una lista de reglas de control de acceso sobre los recursos que puede utilizar cada programa. Se encuentran mayormente estructurados en 3 bloques, las abstracciones, que son otros perfiles utilizados para agrupar reglas y ser reutilizados por los perfiles de AppArmor, las capacidades, las cuales definen los recursos del sistema a los cuales puede acceder la aplicación que utilice el perfil, y en el tercer bloque se encuentran las rutas a las cuales puede acceder la aplicación y cuáles son los permisos con que lo aran. Cada perfil se puede cargar en modo enforce o en modo complain. El modo enforce aplica las reglas y registra las tentativas de violación, mientras que en el modo complain sólo se registran las llamadas al sistema que hubieran sido bloqueadas, pero no se bloquean realmente.

Por ejemplo /etc/apparmor.d/bin.ping es el perfil de AppArmor para la orden /bin/ping la cual realiza la acción de ejecutar el ping del SO (6).

La Figura 1 muestra las características del perfil bin.ping.

```
#include <tunables/global>
/bin/ping flags=(complain) {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,
  network inet raw,

  /bin/ping mixr,
  /etc/modules.conf r,
}
```

Figura 1 Características del perfil bin.ping. Fuente: Elaboración propia

Tabla 1. Contenido del perfil bin.ping. Fuente: Elaboración propia

Contenido del perfil:
#include <tunables/global>
/bin/ping flags=(complain) {
#include <abstractions/base>
#include <abstractions/consoles>

#include <abstractions/nameservice> capability net_raw, capability setuid, network inet raw, /bin/ping mixr, /etc/modules.conf r,	
#include <tunables/global>	Incluye declaraciones de los otros archivos. Esto permite que las declaraciones pertenecientes a varias aplicaciones se puedan colocar en un archivo común.
/bin/ping flags=(complain)	Ruta del programa perfilado, también establece el modo a complain[1].
capability net_raw	Permite a la aplicación acceder a la capacidad CAP_NET_RAW Posix.1e.
/bin/ping mixr	Permite a la aplicación leer y ejecutar el acceso al archivo.

Entradas de ruta o vía: detallan a qué archivos puede acceder una aplicación en el sistema de archivos.

Entradas de capacidad: determinan qué privilegios puede utilizar un proceso confinado. Para añadir o crear un perfil de AppArmor para una aplicación, utiliza un método de creación de perfiles independiente o sistemático, dependiendo de sus necesidades (6).

1.3 Sistemas homólogos

Con el objetivo de verificar las herramientas equivalentes que existen en el mercado a continuación se muestra una caracterización de las mismas.

YaST

YaST es la herramienta de instalación y configuración de openSUSE y la distribución SUSE Linux Enterprise. Es popular por su facilidad de uso y la interfaz gráfica atractiva y la capacidad de personalizar su sistema de forma rápida durante y después de la instalación. YaST es un acrónimo del inglés Yet another Setup Tool lo que se podría traducir como: otra herramienta de configuración. YaST se puede utilizar para configurar todo el sistema. La configuración de

1 modo complain: es el modo en que un perfil se encuentra cuando sobre él se realiza un test y se notifica al sistema si existe alguna violación de seguridad, sin bloquear el funcionamiento.

hardware, configurar la red, los servicios del sistema y ajustar la configuración de seguridad. A todas estas tareas se puede llegar desde el centro de control YaST (7).

YaST proporciona un módulo para añadir, modificar, eliminar y listar los perfiles del sistema. Cuenta con dos interfaces: una gráfica y una basada en texto. La interfaz basada en texto consume menos recursos y ancho de banda, lo que lo convierte en una mejor opción para la administración remota, o cuando un entorno gráfico local es inconveniente. Aunque las interfaces tienen diferentes apariencias, ofrecen la misma funcionalidad de manera similar (8).

aa-autodep

Aa-autodep se utiliza para generar un perfil mínimo de AppArmor para un conjunto de ejecutables, así como programas de escritura interpretados. Esta herramienta proporciona un perfil base que contenga una directiva que incluye las entradas básicas genéricas para las aplicaciones. El perfil es generado por la llamada recursiva a los ejecutables listados en la línea de comandos (9).

Resultado de los sistemas homólogos

Realizando un estudio de las herramientas de gestión de perfiles del *framework* de seguridad AppArmor, el autor de la presente investigación propone no utilizar YaST como herramienta para ser incluida en la distribución cubana GNU/Linux Nova en su variante para servidores. Esto se debe a que la misma está desarrollada para las distribuciones de GNU/Linux OpenSuse y Suse Linux Enterprise, lo que trae consigo el uso de paquetes .rpm y en la distribución cubana se utiliza el sistema de paquetes .deb, lo que la hace incompatible.

En estas distribuciones las direcciones de los ficheros de configuración de los servicios y sus ejecutables cambian con respecto a la distribución Ubuntu de GNU/Linux en la cual actualmente se basa GNU/Linux Nova. YaST está implementada en el lenguaje de programación C, lo que la hace incompatible con la herramienta de administración nova-manager. Sin embargo, sus funcionalidades en conjunto con la herramienta aa-autodep, se tendrán en cuenta para el desarrollo de la solución al problema planteado, en particular serán implementadas las funcionalidades presentes en YaST que se mencionan a continuación:

- Adicionar perfil
- Modificar perfil
- Eliminar perfil
- Listar perfiles

1.4 Metodología y herramientas

Para el desarrollo de las nuevas funcionalidades propuestas para el módulo se define el uso de las siguientes herramientas y metodología, las cuales constituyen un requisito para poder integrar el módulo a la herramienta nova-manager.

1.4.1 Metodología

Una metodología es un proceso que engloba procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software. Se va indicando paso a paso lo que se debe hacer para lograr un producto informático. Además, se deben especificar las personas que van a participar en el proceso, así como el papel que van jugar en él (10). A continuación, se explica la metodología utilizada como guía en la confección del trabajo.

Variación de AUP para la UCI

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas. Estas prácticas se centran en el desarrollo de productos y servicios de calidad (11).

Objetivos de la variación de AUP para la UCI

Modelado de negocio: El modelado del negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Para modelar el negocio se proponen las siguientes variantes:

- Casos de Uso del Negocio (CUN).
- Descripción de Proceso de Negocio (DPN).
- Modelo Conceptual (MC).

Requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no

funcionales del producto. Existen tres formas de encapsular los requisitos, casos de uso del sistema (CUS), historias de usuario (HU) y descripción de requisitos por proceso (DRP), agrupados en cuatro escenarios condicionados por el Modelado de negocio (11).

Escenarios para la disciplina Requisitos

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (CUN, DPN o MC) y existen tres formas de encapsular los requisitos (CUS, HU, DRP), surgen cuatro escenarios para modelar el sistema en los proyectos. Para el desarrollo de la herramienta se utilizó el escenario número 4 dado que no se modela el negocio, se utiliza este escenario para modelar el sistema y los requisitos serán descritos mediante el uso de HU.

1.4.2 Lenguaje de programación

Se empleará como lenguaje de programación Python por su simplicidad, versatilidad y rapidez de desarrollo. Es un lenguaje de scripting independiente de plataforma y orientado a objetos, que está preparado para el desarrollo de aplicaciones destinadas a servicios web. Es un lenguaje interpretado, por lo cual no es necesario compilar el código fuente para que el mismo sea ejecutado, ofreciendo rapidez en el desarrollo. Dispone de diversas funciones brindándole la posibilidad de tratar con cadenas de texto, números y archivos, contando además con librerías externas que podrían ser importadas al proyecto para el trato de temas específicos, como crear archivos comprimidos de extensión .zip (12). Se pretende que la herramienta sea desarrollada en forma de módulo para ser incluido en nova-manager para Nova servidores, el cual cuenta con todos sus módulos desarrollados en Python.

1.4.3 Entorno de desarrollo integrado (IDE por sus siglas en inglés)

El IDE PyCharm se caracteriza por ser multiplataforma, se trabaja en su versión para 2016, desarrollado por la empresa JetBrains. Este IDE contiene mejoras en la gestión de los intérpretes de Python con una nueva interfaz de usuario, para intérpretes remotos y soporte completo de depuración en la consola interactiva de Python (13). Las ventajas diferenciales de esta herramienta sobre otros IDE radica en su integración con GIT[2].para el control de versiones y en la posibilidad de crear plantillas personalizadas para la generación de clases.

1.4.4 Modelado

2 GIT: es una herramienta para mantener el control de versiones de un software que se encuentre en desarrollo.

El Lenguaje de Modelado Unificado (Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reutilizables.

Visual Paradigm en su versión 8.0, se selecciona como herramienta de modelado, utiliza UML para la representación de las etapas por las que trasciende un producto de software. Permite la realización de los diagramas necesarios para el completo modelado de la aplicación, a su vez cuenta con generación de código fuente desde los mismos y la documentación asociada al proceso en actual modelación (14).

Valoraciones del capítulo

- El análisis de las herramientas de gestión de perfiles del *framework* de seguridad AppArmor permitió identificar funcionalidades para implementar durante el desarrollo de la herramienta.
- La herramienta aa-autodep será empleada directamente como utilidad para la construcción de los perfiles de AppArmor.
- Las tecnologías seleccionadas permitirán que al concluir el desarrollo la herramienta de gestión de perfiles de seguridad para AppArmor la misma pueda integrarse a nova-manager.

Capítulo 2: Análisis y diseño

En este capítulo se representará la completa estructura de la herramienta para la gestión de perfiles de AppArmor, dando a conocer las funcionalidades de la aplicación, requisitos funcionales y no funcionales, historias de usuario y diagramas de clases.

2.1 Propuesta del sistema a desarrollar

Luego de concluir el estudio de homólogos realizado en el epígrafe 1.3 se propone desarrollar un módulo para la herramienta nova-manager presente en la distribución GNU/Linux Nova en su variante para servidores. El mismo contará con funcionalidades que le permitirán gestionar los perfiles del *framework* de seguridad, cambiar el modo de ejecución de los mismos, así como habilitarlos o deshabilitarlos. Permitirá además la creación de nuevos perfiles para aplicaciones aún no instaladas en el sistema. El módulo a implementar facilitará la gestión de los perfiles evitando la ocurrencia de errores humanos.

2.1.1 Descripción de los componentes del sistema

A continuación, se muestran los componentes por los cuales estará constituido el sistema que se pretende desarrollar, se brinda una breve descripción y cómo funcionan cada uno de ellos.

- **nova-manager*:*** Gestor de administración de GNU/Linux Nova en su versión para servidores. Se encarga de la instalación, configuración y desinstalación de los servicios para servidores con que cuenta Nova Servidores 6.0. Maneja las validaciones de configuraciones, las utilidades e internacionalización de la herramienta y las vistas con las que interactuarán los usuarios.
- **nova_apparmor***:* módulo para la gestión de perfiles de seguridad del *framework* de seguridad AppArmor.
- **frontend:** paquete donde se encontrarán definidas todas las clases de las vistas de nova_apparmor.
- **backend*:*** es el paquete donde se encontrarán definidas las clases controladoras.
- **locale*:*** paquete de internacionalización.
- ***common:*** se encuentran definidos los modelos de datos y los validadores.

A continuación, se muestra el modelo de relación entre los paquetes del análisis del sistema para GNU/Linux Nova servidores. En el modelo se detallan los principales componentes del sistema y las relaciones que se establecen entre ellos.

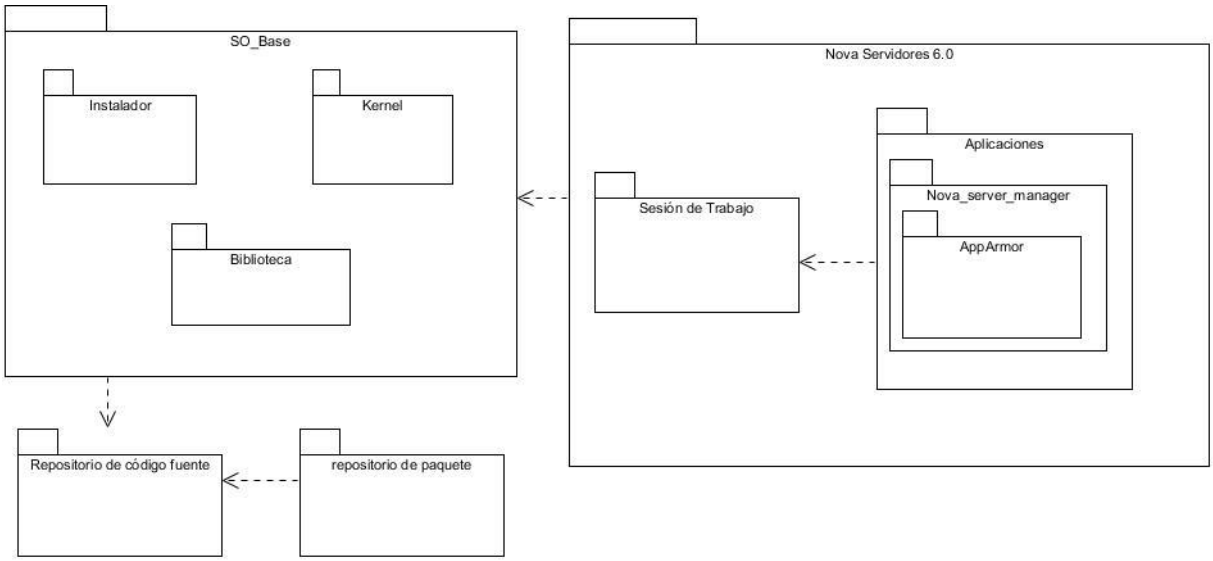


Figura 2. Diagrama de paquetes de análisis del diseño de Nova Servidores 6.0. Fuente: Elaboración propia

A continuación se ilustra el paquete nova_apparmor, mostrando la relación entre los paquetes.

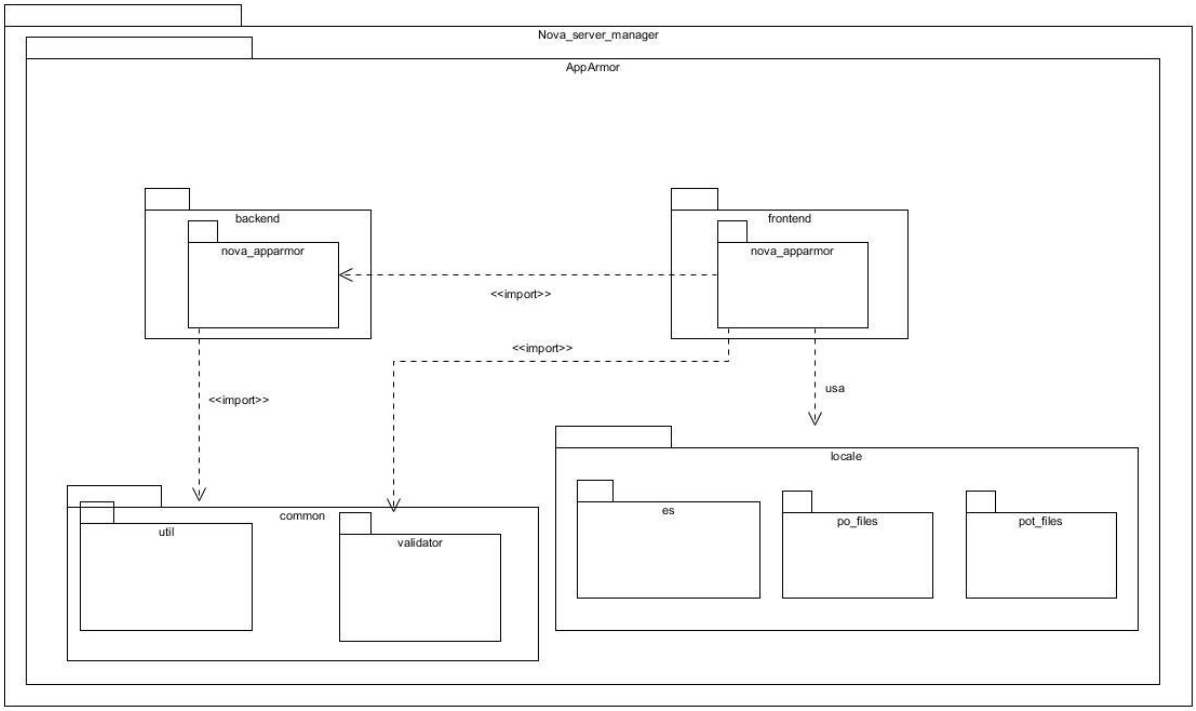


Figura 3. Diagrama de paquetes para el módulo nova_apparmor. Fuente: Elaboración propia

2.2 Requisitos funcionales del sistema

Se identifican a partir de necesidades de negocio o necesidades de los clientes y usuarios. Describen beneficios que recibirán la organización y/o sus clientes con el desarrollo del sistema (11).

Tabla 2. Tabla de requisitos funcionales. Fuente: Elaboración propia

No.	Nombre del requisito	Descripción
RF1	Adicionar perfil	Utiliza la herramienta aa-autodep de *AppArmor* para generar un nuevo perfil de la aplicación deseada, incluyendo en este las dependencias necesarias para el correcto funcionamiento. En caso de que la aplicación no se encuentre en el sistema, se puede crear un perfil nuevo para la aplicación deseada.
RF2	Eliminar perfil	Se encarga de eliminar un perfil seleccionado.
RF3	Modificar perfil	Brindará la posibilidad de modificar perfiles existentes, aumentando o disminuyendo el nivel de seguridad de este, adicionando o eliminado abstracciones, capacidades o rutas con permisos.
RF4	Habilitar perfil	Habilita un perfil que haya sido Deshabilitado anteriormente.
RF5	Deshabilitar perfil	Deshabilita el funcionamiento del perfil deseado, sin eliminarlo.
RF6	Listar Perfiles	Muestra todos los perfiles existentes en el *framework.*
RF7	Aplicar modo Complain	Se selecciona un perfil para ser monitorizado, y registrar las violaciones de las restricciones, pero sin restringir el acceso.
RF8	Aplicar modo Enforce	Se selecciona un perfil para ser monitorizado, y registrar las violaciones de las restricciones, denegando el acceso al recurso y la aplicación

		perteneciente al perfil se confina.
RF9	Listar aplicaciones con conexiones tcp-udp activas no confinadas y confinadas	Lista las aplicaciones con conexiones tcp-udp activas que no están confinadas y las confinadas.
RF10	Instalar servicio	Se encarga de instalar apparmor-utils en caso de no se encontrarse en el sistema.
RF11	Desinstalar servicio	Se encarga de desinstalar apparmor-utils del sistema en caso de que el usuario lo desee.
RF12	Iniciar servicio	Inicia AppArmor.
RF13	Detener servicio	Detiene AppArmor.
RF14	Reiniciar servicio	Reinicia AppArmor.

2.3 Requisitos no funcionales del sistema

Son restricciones de los servicios o funciones ofrecidas por la solución informática (tiempo, proceso o estándares del dominio de negocio), además de cualidades que se le imponen al sistema en cuanto al diseño y la implementación. Normalmente afectan al sistema en su totalidad más que una característica o servicio en particular (11).

Requerimientos de software:

RNF1: Se requiere de la distribución de GNU/Linux Nova.

RNF2: Se requiere la instalación del paquete apparmor-utils.

Requerimientos de hardware:

RNF3: PC con 512MB de memoria RAM o superior.

Requerimientos de diseño e implementación:

RNF4: Utilizar como lenguaje de programación Python.

RNF6: El uso de herramientas y recursos libres, los cuales se podrán usar, modificar y distribuir libremente.

2.4 Historias de usuario

La metodología empleada propone tres variantes a utilizar en los proyectos (CUN, DPN o MC) de esta manera surgen cuatro escenarios para modelar el sistema en los proyectos, manteniendo en dos de ellos MC (11). Para el desarrollo de la solución propuesta, se cuenta con un negocio muy bien definido y con la compañía del cliente junto al equipo de desarrollo para precisar detalles acerca de los requisitos para la implementación, prueba y validación. Teniendo en cuenta lo planteado se decide adoptar por el escenario número cuatro en el cual serán usadas las historias de usuarios (HU) para encapsular los requisitos.

Las HU utilizadas en las metodologías de desarrollo ágiles para especificar los requisitos, constituyen una forma de administrar los requisitos sin la necesidad de una gran cantidad de documentación y pueden ser elaboradas en un corto período de tiempo para administrarlos. Las mismas se describen mediante la utilización del lenguaje común del usuario facilitando su redacción (15).

Se define una HU para cada uno de los requisitos funcionales para un total de 14 HU. A continuación, se muestran las HU “Adicionar perfil”, “Modificar perfil”, “Listar perfiles” y “Habilitar perfil”. El resto de las historias de usuarios se encuentran definidas en el Anexo 1.

Tabla 3. Historia de usuario para el requisito adicionar perfil. Fuente: Elaboración propia

Historia de Usuario	
Numero: HU_1	**Nombre:** Adicionar perfil
Prioridad de negocio: Alta	**Iteración asignada:** 1
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 2 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 4 días
Descripción: Utiliza la herramienta aa-autodep de *AppArmor* para generar un nuevo perfil de la aplicación deseada, incluyendo en este las dependencias necesarias para el correcto funcionamiento. En caso de que la aplicación no se encuentre en el sistema se puede crear un perfil nuevo para la aplicación que deseemos.	
Prototipo:	

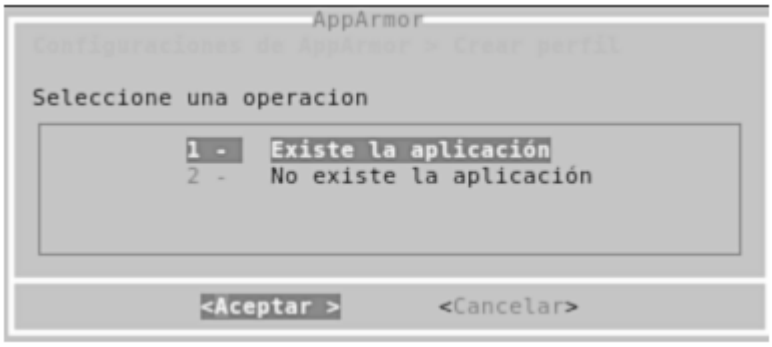


Figura 4. Seleccionar tipo de perfil a crear

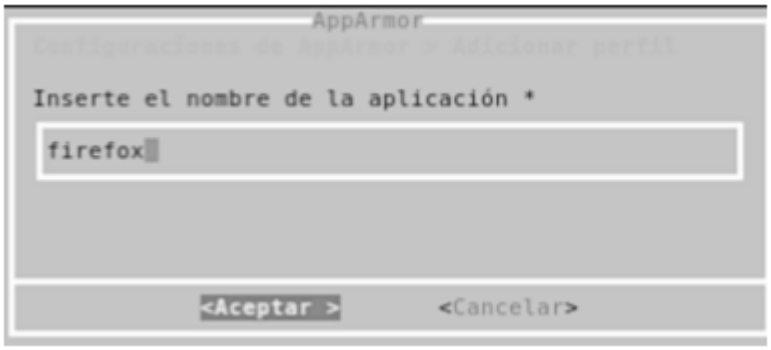


Figura 5. Crear perfil nuevo a partir de una aplicación instalada

Figura 6. Crear perfil nuevo sin contar con la aplicación

Tabla 4 Historia de usuario para el requisito modificar perfil. Fuente: Elaboración propia

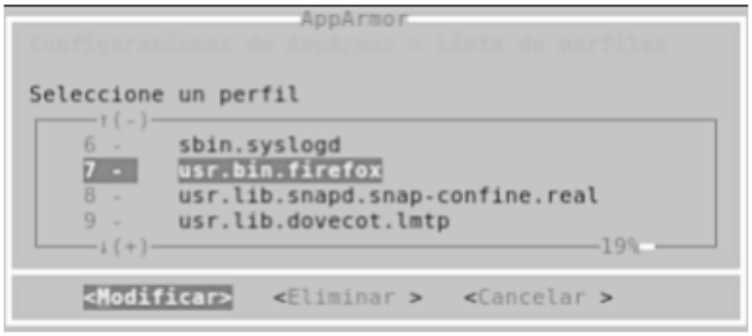
Historia de Usuario	
Numero: HU_3	**Nombre:** Modificar perfil
Prioridad de negocio: Alta	**Iteración asignada:** 5
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 4 días
Riesgo en Desarrollo: - Interrupción del fluido eléctrico	**Tiempo real:** 5 días
Descripción: Brindará la posibilidad de modificar perfiles existentes, aumentando o disminuyendo el nivel de seguridad de este, adicionando o eliminando abstracciones, capacidades o rutas con permisos.	
Prototipo: 	

Figura 7. Seleccionar perfil para modificar

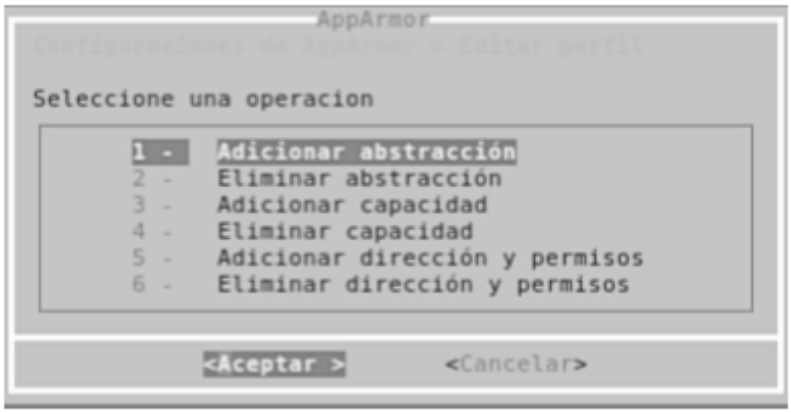


Figura 8. Características para ser editadas en el perfil

Tabla 5 Historia de usuario para el requisito listar perfiles. Fuente: Elaboración propia

Historia de Usuario	
Numero: HU_5	**Nombre:** Listar perfiles
Prioridad de negocio: Alta	**Iteración asignada:** 5
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 2 días
Riesgo en Desarrollo: - Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Se muestra una lista de todos los perfiles existentes en el sistema brindando a su vez en la vista la opción de ser seleccionado el perfil deseado para ser modificado o eliminado.	
Prototipo:  	

Figura 9. Listado de todos los perfiles

Tabla 6 Historia de usuario para el requisito habilitar y deshabilitar perfil. Fuente: Elaboración propia

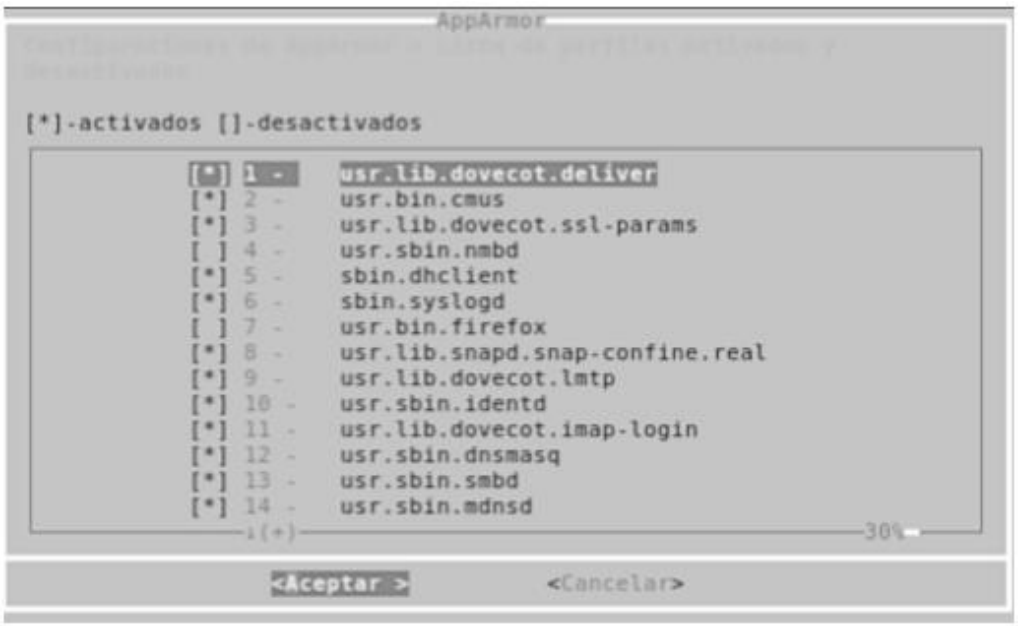
Historia de Usuario	
Numero: HU_4	**Nombre:** Habilitar perfil
Prioridad de negocio: Alta	**Iteración asignada:** 4
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 día
Riesgo en desarrollo: - Interrupción del fluido eléctrico	**Tiempo real:** 1 día
Descripción: Se encarga de poner en funcionamiento un perfil seleccionado que haya sido deshabilitado anteriormente o se encuentre deshabilitado cuando sea instalado el servicio de A*ppArmor*.	
Prototipo: 	

Figura 10. Perfiles habilitados y deshabilitados

2.5 Diseño

El diseño en el ciclo de vida de un software facilita la comprensión de su funcionamiento y brinda una representación o modelo del mismo, con el fin de definir detalles necesarios para permitir su realización física. El modelo de diseño cuenta con una representación arquitectónica del software que sirve de punto de partida para trazar las tareas de implementación, y así llegar de forma correcta a los requisitos del sistema (16).

2.5.1 Descripción de la arquitectura del software

Según Roger S. Pressman: “En su forma más simple, la arquitectura del software es la estructura u organización de los componentes del programa, la manera en que estos interactúan y la estructura

de datos que utilizan" (16). Consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. Es considerada el nivel más alto en el diseño de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software (17).

El Estilo Modelo Vista Controlador (MVC) está basado en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Permite la separación de los datos de una aplicación, la interfaz de usuario y la lógica de control, en tres capas: el modelo, la vista y el controlador (18).

Modelo: Encargado de manejar los datos asociados al sistema y las operaciones asociadas a esos datos, todo lo referente a la persistencia de datos de la aplicación, las clases entidades, el acceso a la red y los elementos necesarios para manejar dichos elementos.

Vista: Define y gestiona como se presentan esos datos al usuario.

Controlador: Dirige la interacción del usuario y pasa estas interacciones a Vista y Modelo. Maneja las solicitudes de eventos de los usuarios y registra los cambios realizados por los mismos.

La implementación de la herramienta será en forma de módulo con el nombre nova_apparmor, estará presente en la aplicación nova-manager incluida al SO GNU/Linux Nova servidores, la cual cuenta con MVC definido para todos sus módulos, mantendrá el estilo en el desarrollo de la aplicación, manteniendo la homogeneidad con todos los módulos existentes en la herramienta.

El módulo contara con los tres componentes, definidos de la siguiente manera:

- Vista: la vista se encontrara en el paquete frontend, el cual se encargara crear las vistas que serán mostradas al usuario, obteniendo la información necesaria para enviarla al controlador.
- Controlador: el controlador se encontrara en el paquete backend, el cual contiene las clases asociadas al trabajo con los perfiles, el controlador se comunica con la vista, la cual le brinda la información que desea conocer el usuario, y haciendo uso del modelo se encarga de entregarle a la vista la información pedida por el usuario.
- Modelo: el modelo se encontrara en la clase File presente en el paquete backend, la cual contiene los métodos para realizar el trabajo con los perfiles, ya que al realizarse el manejo de los datos mediante la persistencia de información en los perfiles, la clase realiza el manejo de la información que contienen los perfiles.

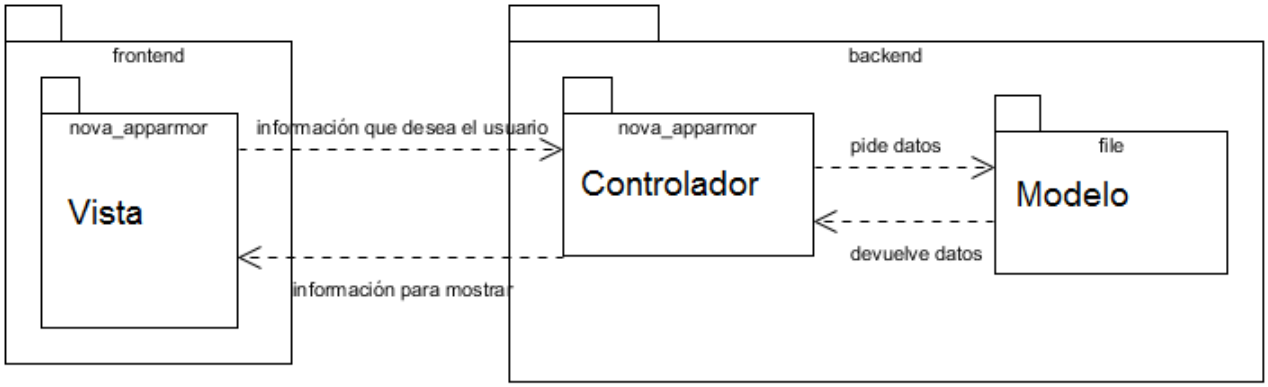


Figura 11. Estilo MVC (modelo-vista-controlado). Fuente: Elaboración propia

2.5.2 Patrones de diseño

Un patrón de diseño se caracteriza como: “una regla de tres partes que expresa una relación entre cierto contexto, un problema y una solución”. El contexto le permite al lector entender de manera más concreta el problema y la solución más apropiada. Un conjunto de requerimientos, incluidas limitaciones y restricciones, actúan como una fuerza que influyen en la manera en la que puede interpretarse el problema en este contexto y en cómo podría aplicarse con eficacia la solución (16).

Bajo acoplamiento: El acoplamiento mide el grado en que una clase está conectada a otra, tiene conocimiento de otra o, de alguna manera, depende de otra. El bajo acoplamiento para lograr una alta reutilización es logrado asignando la responsabilidad de manera que el acoplamiento permanezca bajo (19).

Desventajas del acoplamiento:

- Los cambios en una clase pueden implicar cambios en las clases relacionadas.
- Dificultad de comprensión.
- Dificultad de reutilización

Consideraciones:

- El bajo acoplamiento permite crear clases más independientes, más reutilizables, lo que implica mayor productividad.
- El acoplamiento puede no ser importante si la reutilización no está en nuestros objetivos.

La presencia del patrón se encuentra en la clase Main en el paquete frontend donde se genera la interfaz y la implementación de los métodos funcionales se realiza en el paquete backend en su clase Main.

```
from backend.nova_apparmor import nova_apparmor


class Main:
    def __init__(self):
        self.backend = nova_apparmor.Main()
```

Figura 12. Bajo acoplamiento en el código. Fuente: Elaboración propia

Alta cohesión: La cohesión mide el grado en que están relacionadas las responsabilidades de una clase. Se aplica para realizar un diseño que evite contener clases con un alto grado de abstracción, delegando las responsabilidades muy complejas a otros objetos. El uso del patrón en la herramienta se evidencia con las validaciones de las vistas, las cuales se encuentran implementadas por separadas de las vistas, en caso de ocurrir cambios sobre las vistas, no se afecta el funcionamiento de las validaciones.

```
from common.validators import generic
from common.validators import file_system


codeapp, app = self.dialog_manager.input(
    _('AppArmor'),
    _('\\Zb\\Z4ApppArmor Configurations > Add Profile\\Zn\n\nInsert the application name'),
    validators=[generic.isNotEmpty],
    required='False')
```

Figura 13. Alta cohesión en el código. Fuente: Elaboración propia

Desventajas de una clase con baja cohesión:

- Difícil de comprender
- Difícil de reutilizar
- Difícil de mantener
- Delicada, se afecta mucho por los cambios

2.5.3 Diagrama de clases de diseño del módulo nova_apparmor:

Un diagrama de clases de diseño es utilizado durante el proceso de análisis y diseño del software, creando un diseño conceptual de la información que se manejará en el software y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. En estos se representan gráficamente las especificaciones de las clases de la aplicación y de las interfaces. En su diseño contiene información sobre clases, métodos, atributos y dependencias (16).

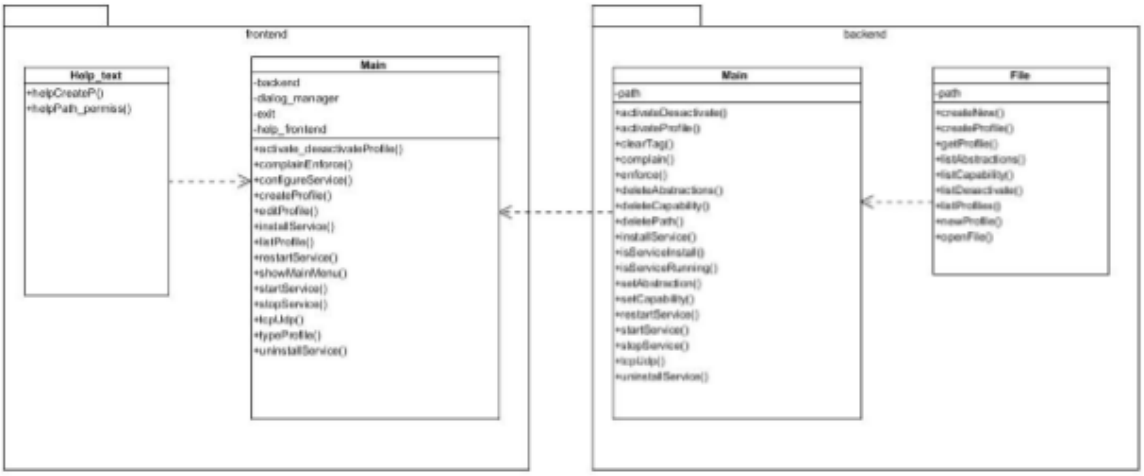


Figura 14.Diagrama de clases del diseño. Fuente: Elaboración propia

Valoraciones del capítulo

- La identificación y documentación de los requisitos funcionales y no funcionales sirvió de guía para la implementación del módulo.
- La definición del estilo Modelo Vista Controlador como arquitectura para el desarrollo del software, siendo esta la arquitectura presente en la herramienta nova-manager para el desarrollo de cada uno de sus presentes módulos, permitirá una completa homogeneidad en dicha herramienta.
- El empleo de los patrones de diseño GRASP permite una solución con objetos que se valgan de la información requerida y definiciones de clases sencillas más fáciles de comprender y de mantener con mejores oportunidades de reutilización.

Capítulo 3: Implementación y prueba

El presente capítulo está enfocado en la descripción del proceso de construcción de la herramienta para la gestión de perfiles de AppArmor desarrollada a través del módulo nova_apparmor para la herramienta nova-manager, además de las pruebas realizadas a los mismos. Se exponen los estándares de codificación empleados para el desarrollo del módulo nova_apparmor y la estrategia de pruebas para comprobar la calidad del módulo.

3.1 Estándar de codificación

Una buena práctica de programación que potencia la legibilidad del código es la utilización de estándares de codificación. Un estándar de codificación comprende todos los aspectos de la generación de código y la legibilidad del mismo.

A continuación, se mencionan las prácticas utilizadas en la implementación del módulo.

Identación: Emplear 4 espacios por cada nivel de identación.

```
# activar perfil sleccionado
def activateProfile(self,n):
    subprocessCommand('rm /etc/apparmor.d/disable/'+n)
    subprocessCommand('cat /etc/apparmor.d/' + n + ' | sudo apparmor_parser -a')
```

Figura 15. Identación. Fuente: Elaboración propia

Tabuladores y espacios: No mezclar tabulaciones con espacios.

```
# devuelve la lista de perfiles
def listProfiles(self):
    _file = []
    l = os.listdir('/etc/apparmor.d/')
    for i in l:
        if os.path.isfile(i):
            _file.append(i)
    return _file
```

Figura 16. Tabuladores y espacios. Fuente: Elaboración propia

Importaciones: Las importaciones deben ser en diferentes líneas.

```
import os.path
from common.util.commands import subprocessCommand
from common.util.commands import subprocessCommandReal
```

Figura 17. Importaciones. Fuente: Elaboración propia

Declaraciones: Se debe declarar cada variable en una línea diferente y de esta forma se puede comentar cada variable por separado.

```
# lista con los nombre de los perfiles desactivados
def profilesDesactivates(self):
    # Lista de perfiles desactivados
    _file = []
    l = os.listdir('/etc/apparmor.d/disable/')
    for i in l:
        if os.path.isfile(i):
            _file.append(i)
    return _file
```

Figura 18. Declaraciones. Fuente: Elaboración propia

Comentarios: Los comentarios deben especificar exactamente la funcionalidad del código.

```
# contenido de un perfil seleccionado
def openFile(self, p):...

# lista de abstracciones
def listAbstractions(self):...
```

Figura 19. Comentarios. *Fuente: Elaboración propia*

Líneas en blanco: Se debe dejar un solo espacio entre cada definición de método.

```
# aplicar modo complain
def complain(self,p):...

# aplicar modo enforce
def enforce(self,p):...
```

Figura 20. Líneas en blanco. Fuente: Elaboración propia

Asignación de nombres: Para nombrar funciones y variables la primera palabra debe ser en minúscula, en caso de ser un nombre compuesto la segunda se inicia en mayúscula.

- Para nombres simples:

```
# aplicar modo complain
def complain(self,p):...
```

Figura 21. Asignación de nombre para nombres simples. *Fuente: Elaboración propia*

- Para nombre compuestos:

```
# instalar los servicios necesarios
def installService(self):
    subprocessCommand('apt-get update')
    return subprocessCommandReal('apt-get install apparmor apparmor-utils --yes')
```

Figura 22. Asignación de nombre para nombres compuestos. Fuente: Elaboración propia

3.2 Empaquetado de la herramienta

A continuación, se describen los pasos que se llevaron a cabo para incorporar la herramienta para la gestión de los perfiles de seguridad del *framework* AppArmor.

Paso 1: descomprimir el SO en una ruta conocida del sistema de ficheros.

Paso 2: acceder a la carpeta install.

Paso 3: dentro de ella se encuentra el archivo *filesystem.squashfs*, dentro del cual se encuentra el sistema operativo base.

Paso 4: se procede a desempaquetar el archivo *filesystem.squashfs*, este paso se describe en 2 pasos:

- Se accede a través de la consola del sistema a la ruta donde se encuentra el archivo, escribiendo en la consola *cd [ruta hasta el archivo]*, ejemplo: *cd /home/miuser/misistema/install.*
- Después se escribe *sudo unsquashfs -d [carpeta_con_mis_archivos]/ -f*

filesystem.squashfs.

Paso 5: luego se accede a la carpeta donde se encuentra el contenido de *filesystem.squashfs* y dentro de ella, en la ruta */usr/share/nova-manager/* se encuentra la herramienta presente en el SO de GNU/Linux Nova.

Paso 6: ya con la ruta de nova-manager se procede a incluir el módulo de AppArmor.

Paso 7: luego de incluido el módulo, se procede a generar un nuevo archivo *filesystem.squashfs* con el módulo de AppArmor, regresando a la carpeta install, donde se encuentra la carpeta *carpeta_con_mis_archivos*, se ejecuta en la consola *mksquashfs carpeta_con_mis_archivos/ filesystem.squashfs.*

Paso 8: después de creado el archivo *filesystem.squashfs* con el módulo dentro, se empaqueta el SO. Ingresando en la consola, *mkisofs -r -V "Nova para Servidores" -cache-inodes -J -l -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -o [ruta_para_el_nuevo_iso]/nuevo.iso [dirección donde se encuentra el sistema descomprimido con el módulo incluido]*

Paso 9: instalar el SO nuevo que ha sido creado.

Todos los comandos anteriormente mencionados han sido ejecutados y probados en GNU/Linux Nova para escritorio y Ubuntu de GNU/Linux en la versión 16.04 en su variante para escritorio.

3.3 Pruebas de software

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan el desempeño y la calidad de un software, así como evaluar los resultados. Las técnicas son variadas y se realizan desde el personal de prueba hasta herramientas automatizadas que para facilitar el trabajo y el rango de tiempo en que se realizan estas (16).

3.3.1 Métodos de prueba

Caja blanca

La prueba de caja blanca, es un método de diseño que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba. Del empleo de sus métodos de prueba, se podrán derivar casos de prueba que (16):

- Garanticen que todas las rutas independientes dentro del módulo se han ejercitado al menos una vez.
- Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.
- Ejerciten los lados verdaderos y falsos de las secciones lógicas.
- Ejerciten estructuras de datos internos para asegurar su validez.

Caja negra

Las pruebas de caja negra, también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software. Es decir, permiten derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de la herramienta. No es una opción frente a las técnicas de caja blanca. Es, en cambio, un enfoque complementario que tiene probabilidades de descubrir errores diferentes de los que se descubrirán con los métodos de caja blanca.

A diferencia de las pruebas de caja blanca, que se realizan al inicio del proceso de prueba, las de caja negra mayormente se aplican durante las últimas etapas de la prueba. Debido a que estas desatienden a propósito la estructura de control, la atención se concentra en el dominio de la información. Las pruebas están diseñadas para responder las siguientes preguntas (17):

- ¿Cómo se prueba la validez funcional?
- ¿Cómo se prueba el comportamiento y el desempeño del sistema?
- ¿Cuáles clases de entrada serán buenos casos de prueba?
- ¿El sistema es particularmente sensible a ciertos valores de entrada?
- ¿Cómo se aíslan los límites de una clase de datos?
- ¿Cuáles tasas de datos y cuál volumen tolera el sistema?
- ¿Qué efectos tienen combinaciones específicas de datos sobre la operación del sistema?

Con la aplicación de la prueba se pretende encontrar:

- Errores en estructuras de datos o en accesos a bases de datos externas.
- Funciones incorrectas o ausentes.
- Errores de inicialización y de terminación.
- Errores en la interfaz.
- Errores de comportamiento o desempeño.

3.3.2 Pruebas a realizar

Unitarias

Las pruebas de unidad se concentran en el esfuerzo de verificación de la unidad más pequeña del diseño del software: el componente o módulo de software. Tomando como guía la descripción del diseño al nivel de componentes, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y errores que estas descubren. Las pruebas de unidad se concentran en la lógica del procesamiento interno y en las estructuras de datos dentro de los límites de un componente. Este tipo de prueba se puede aplicar en paralelo a varios componentes (16).

Funcionales

Las pruebas de sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica (16).

Integración

La prueba de integración es una técnica sistemática para construir la arquitectura del software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño.

Se selecciona como estrategia para realizar la prueba, la integración ascendente, la cual según Pressman describe:

La prueba de integración ascendente, como su nombre lo indica, empieza la construcción y la prueba con módulos atómicos (es decir, componentes de los niveles más bajos de la estructura del programa). Debido a que los componentes se integran de abajo hacia arriba, siempre está

disponible el procesamiento requerido para los componentes subordinados a un determinado nivel y se elimina la necesidad de resguardos (16).

Una estrategia de integración ascendente se implementa mediante los siguientes pasos:

- Se combinan los módulos de bajo nivel en grupos (también llamados construcciones) que realicen una subfunción específica del software.
- Se escribe un controlador (un programa de control para pruebas) con el fin de coordinar la entrada y la salida de los casos de prueba.
- Se prueba el grupo.
- Se eliminan los controladores y se combinan los grupos ascendiendo por la estructura del programa.

3.3.3 Técnicas de prueba

Camino básico

La prueba de camino básico es una técnica de prueba de caja blanca. Este método permite que el diseñador de casos de pruebas obtenga una medida de complejidad lógica de un diseño procedimental y que use esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para ejercitar el conjunto básico deben garantizar que se ejecuta cada instrucción del programa por lo menos una vez durante la prueba (16).

Partición equivalente

La partición equivalente es un método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. Un caso de prueba ideal de manejo simple descubre una clase de errores (por ejemplo, procesamiento incorrecto de todos los datos de caracteres) que, de otra manera, requerirá la ejecución de muchos casos antes de que se observe el error general. La partición equivalente se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse.

Según Pressman las clases de equivalencia se definen de acuerdo con las siguientes directrices:

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
2. Si una condición de entrada requiere un valor especifico, se define una clase de

equivalencia válida y dos no válidas.

3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y otras no válidas.
4. Si una condición de entrada es booleana, se define una clase de equivalencia válida y otras no válidas.

Al aplicar estas directrices para la derivación de clases de equivalencia, se desarrollarán y ejecutarán los casos de prueba para cada objeto de los datos del dominio de entrada. Los casos de prueba se seleccionan de modo que el mayor número de atributos de clase de equivalencia se ejercita una vez (16).

3.4 Aplicación de las pruebas

3.4.1 Pruebas unitarias

Se realizarán las pruebas unitarias utilizando el método de caja blanca con la técnica camino básico, al método deletePath() en la clase *Main* en el paquete *backend.*

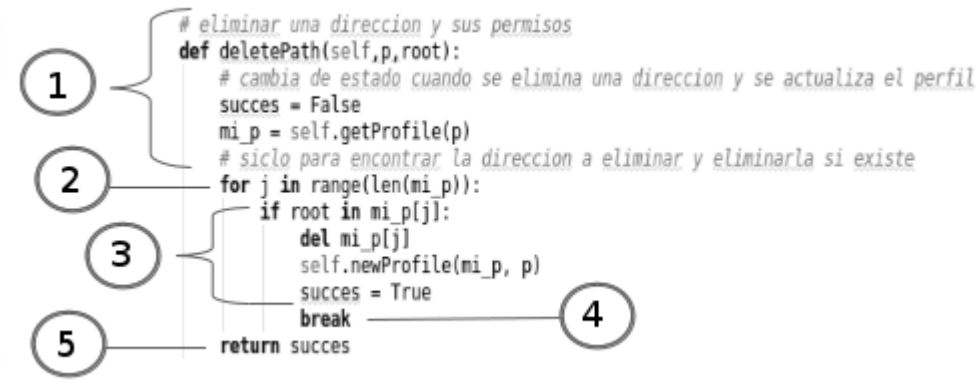


Figura 23. Método deletePath(). Fuente: Elaboración propia

Luego de numerar las líneas de código, se diseña la gráfica del programa que describe el flujo de control lógico empleando nodos y aristas.

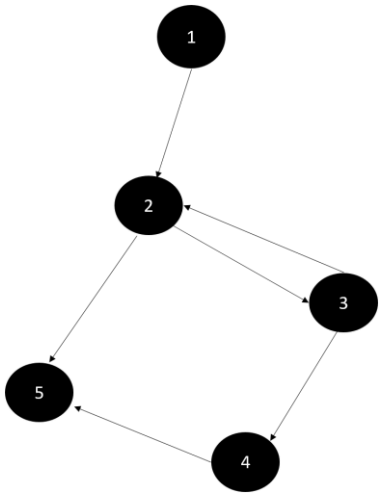


Figura 24. Grafo de flujo. Fuente: Elaboración propia

A partir del grafo obtenido con 5 nodos y 6 aristas, se calcula la complejidad dicromática *V(G)* la cual constituye una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.

V(G) = cantidad_de_aristas – cantidad_de_nodos + 2

V(G) = 6 – 5 + 2 = 3

El valor de V(G) indica el número de rutas linealmente independientes de la estructura de control del programa. En el caso del procedimiento promedio se espera especificarse tres caminos:

No	Caminos
1	1-2-5
2	1-2-3-2-5
3	1-2-3-4-5

El valor de la complejidad ciclomática ofrece además un límite superior para la cantidad de pruebas que se deben diseñar y ejecutar para garantizar que se cumplen todas las sentencias del programa (16). A continuación, se muestran los casos de pruebas para cada camino independiente.

Tabla 7 Caso de prueba para el camino 1-2-5 y 1-2-3-2-5. Fuente: Elaboración propia

Caso de prueba de unidad	
No. camino: 1	**Camino:** 1-2-5
Nombre de la persona que realiza la prueba: Asney Hidalgo Palmero	
Descripción de la prueba: Eliminar la ruta ingresada por el usuario en el perfil seleccionado	
Entrada: Datos del perfil	
Resultado esperado: Mostrar un mensaje informando que no existió cambio en el perfil seleccionado	
Evaluación de la prueba: Resultado satisfactorio, se mostró el mensaje correctamente.	

Caso de prueba de unidad	
No. camino: 2	**Camino:** 1-2-3-2-5
Nombre de la persona que realiza la prueba: Asney Hidalgo Palmero	
Descripción de la prueba: Eliminar la ruta ingresada por el usuario en el perfil seleccionado	
Entrada: Datos del perfil y dirección para ser eliminada del perfil	
Resultado esperado: Mostrar un mensaje informando que la ruta que se desea eliminar no se encuentra	
Evaluación de la prueba: Resultado satisfactorio, se mostró el mensaje correctamente	

Tabla 8. Caso de prueba para el camino 1-2-3-4-5. Fuente: Elaboración propia

Caso de prueba de unidad	
No. camino: 3	**Camino:** 1-2-3-4-5
Nombre de la persona que realiza la prueba: Asney Hidalgo Palmero	
Descripción de la prueba: Eliminar la ruta ingresada por el usuario en el perfil seleccionado	
Entrada: Datos del perfil y dirección para ser eliminada del perfil	
Resultado esperado: Mostrar un mensaje informando que el perfil ha sido actualizado	
Evaluación de la prueba: Resultado satisfactorio, se mostró el mensaje correctamente	

Resultado de las pruebas unitarias

Fueron realizadas tres iteraciones mediante el método de caja blanca utilizando la técnica de camino básico, se encontraron dos errores de validación en el momento de eliminar la ruta en el perfil.

3.4.2 Pruebas funcionales

Las pruebas funcionales fueron realizadas mediante en el método de caja negra utilizando la técnica partición equivalente. A continuación, se muestra el diseño de Casos de Pruebas basados en Historias de Usuarios para el caso de prueba “Adicionar perfil”.

Caso de prueba: Adicionar perfil

Descripción de las variables

Tabla 9. Descripción de las variables del caso de prueba "Adicionar perfil". Fuente: Elaboración propia

No	**Nombre de Campo**	**Clasificación**	**Valor Nulo**	**Descripción**
1	Introduzca nombre de la aplicación.	Cadena	No	Debe ser una aplicación instalada en el sistema operativo, no bebe estar vacío.

Tabla 10. Descripción de los escenarios del caso de prueba " Adicionar perfil". Fuente: Elaboración propia

Escenario	Descripción	Nombre de la aplicación	Respuesta del sistema	Flujo central
EC 1.1 Crear perfil para la aplicación instalada.	Se crea el perfil para la aplicación con las configuraciones necesarias.	rsyslog	El sistema muestra un mensaje de confirmación “el perfil ha sido creado satisfactoriamente.”	1. El usuario selecciona la opción gestionar servicio y presiona aceptar. 2. El usuario selecciona la opción Adicionar perfil y presiona aceptar. 3. El usuario selecciona la opción crear perfil para aplicación instalada y presiona aceptar. 4. El usuario introduce el nombre de la aplicación y presiona aceptar.
EC 1.2 Crear perfil para la aplicación instalada con campos incorrectos.	No se crea el perfil de seguridad.	asd	El sistema muestra un mensaje “La aplicación no se encuentra instalada en el sistema”	

Resultados obtenidos de los casos de pruebas

Se utilizó el método de caja negra para realizar las pruebas sobre la interfaz del módulo nova_apparmor para la herramienta nova-manager. Donde fueron encontradas 5 no conformidades en 2 iteraciones. De estas no conformidades 4 fueron de errores ortográficos y 1 de validación incorrecta. En la siguiente gráfica se muestra las iteraciones realizadas y las no conformidades detectadas, las resueltas y las pendientes.

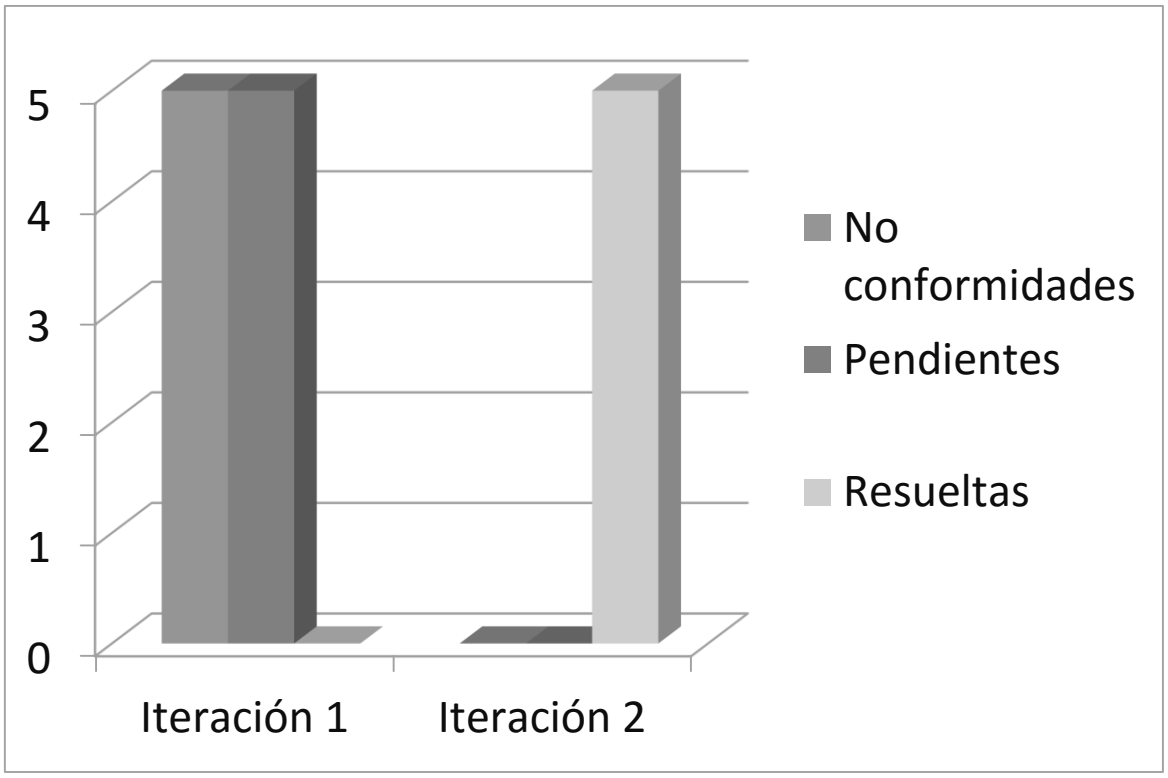


Figura 25. Estadísticas de las pruebas funcionales. Fuente: Elaboración propia

3.4.3 Pruebas de integración

Las pruebas de integración fueron realizadas utilizando la técnica de integración ascendente. En la siguiente tabla se muestran los componentes en nivel ascendente y la integración con el componente en el nivel superior.

Tabla 11. Tabla de niveles de las pruebas de integración

Nivel	Componente	Descripción	Integración	Resultado
1	Módulo AppArmor	Módulo para la gestión de los perfiles del *framework* de seguridad AppArmor para GNU/Linux.	Herramienta nova-manager	Satisfactorio
2	Herramienta nova-manager	Herramienta de administración para Nova en su versión para servidores. Se encarga de la administración de los servicios para servidores presentes en Nova Servidores 6.0.	Personalización de la distribución de GNU/Linux Nova Servidores 6.0	Satisfactorio

Resultado obtenido en las pruebas de integración

Mediante las pruebas de integración de forma ascendente se pudo comprobar la integración de los componentes de la solución donde los resultados obtenidos fueron satisfactorios.

3.4.4 Pruebas de aceptación

Con la aplicación de esta prueba se validará la aceptación del cliente de acuerdo al sistema desarrollado, para de esta manera se haga constar la conformidad del mismo. Escogiendo las pruebas alfas las cuales ser realizarán por parte del cliente en un entorno controlado usando el software en su forma natural, donde el desarrollador del sistema estará presente como observador del usuario, todo el proceso se realizará en un ambiente equivalente a donde será implantado el sistema en uso del cliente. Logrando llevar a cabo todo el proceso, se procederá a efectuar las pruebas y se documentarán los resultados.

Se realizó la prueba de aceptación con el cliente, la cual concluyó que fueron cumplidas todas las funcionalidades, satisfaciendo las necesidades del mismo, se emitió el acta de aceptación el cual se encuentra presente en el Anexo 2.

Valoraciones del capítulo

- Con el uso del estándar de codificación seleccionado se logró obtener un código más robusto y legible.
- Las pruebas realizadas, utilizando los métodos de caja blanca y caja negra, mediante el uso de las técnicas camino básico y partición equivalente, permitieron obtener un módulo con una correcta interfaz y el código en un estado adecuado a las necesidades del cliente.

Conclusiones generales

La realización de la presente investigación permitió desarrollar un módulo para la herramienta nova-manager que gestiona los perfiles del *framework* de seguridad AppArmor del sistema operativo GNU/Linux Nova servidores, dando cumplimiento al objetivo propuesto.

Al terminar la investigación se concluye:

- El análisis de las herramientas de gestión de perfiles del *framework* de seguridad AppArmor posibilitó identificar funcionalidades para implementar durante el desarrollo de la herramienta, seleccionando aa-autodep como herramienta utilitaria para la creación de perfiles.
- Con la utilización del estilo MVC se logró obtener un software capaz de realizar una correcta separación de responsabilidades en el código, a la vez que se facillitó la integración con la herramienta nova-manager.
- A partir de la aplicación de las pruebas se logró comprobar el correcto funcionamiento del módulo, y mediante la realización de pruebas de aceptación el cliente mostró su conformidad con la solución desarrollada.

Recomendaciones

Para complementar la investigación se recomienda:

- Implementar un sistema de notificaciones para el módulo que se encargue de informar los intentos de violación bloqueados por el *framework.*

Referencias bibliográficas

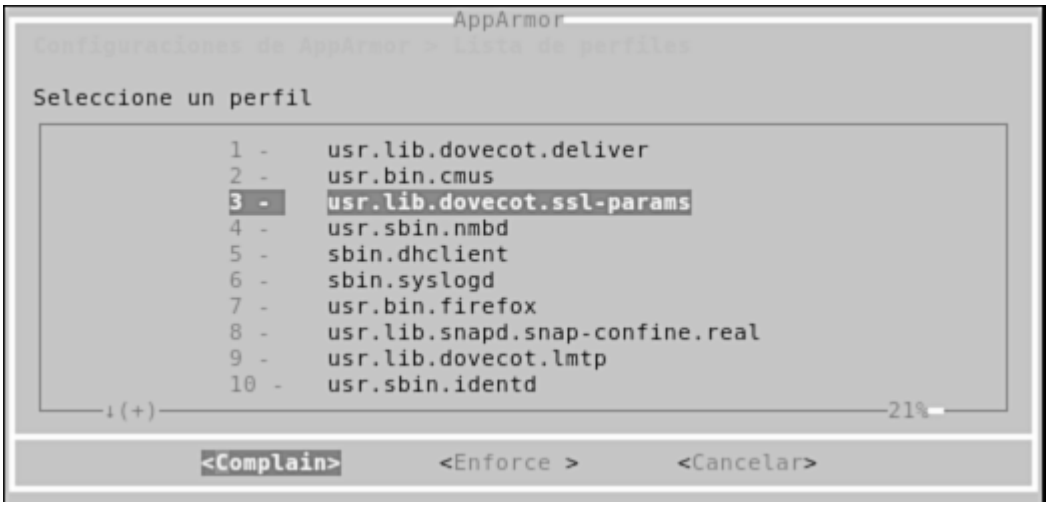
1. ***EL sistema operativo GNU. 2017.*** *¿Qué es el software libre? [En línea] [Citado el: 20 de Enero de 2017.] http://www.gnu.org*
2. **Emc2Net**. **2016 - 2017** Software Libre. [En línea] [Citado el: 23 de Enero de 2017.] https://e-mc2.net/es/linux-es
3. **Dairon Vera Rodriguez**. **2016**. Framework de seguridad. [Citado el: 21 de Noviembre de 2016.]
4. **The server labs. 2005 - 2017.** Framework de Seguridad. [En línea] [Citado el: 10 de Noviembre de 2017.] http://www.theserverlabs.com/es/soluciones/framework-de-seguridad.html
5. **Josep Jorba Esteve. 2004.** Administración avanzada de GNU/Linux. [En línea] [Citado el: 10 de Noviembre de 2016.] http://www.utic.edu.py/citil/images/Manuales/Admin_GNULinux.pdf
6. **Mini-Manual de configuración de marco de seguridad informática Apparmor**. [En línea] [Citado el: 21 de Enero de 2017.] http://humanos.uci.cu/2011/06/un-acercamiento-a-los-frameworks-de-seguridad-informaticafinal/
7. **OpenSUSE. 2011.** YaST. [En línea] [Citado el: 5 de Junio de 2017.] https://es.opensuse.org/Portal:YaST.
8. **OpenSUSE. 2011.** Creación y administración de perfiles con YaST. [En línea] [Citado el: 6 de Junio de 2017.] https://doc.opensuse.org/documentation/leap/security/html/book.security/cha.apparmor.yast.html
9. **Ubuntu Manpage. 2010.** aa-autodep. [En línea] [Citado el: 6 de Junio de 2017.] http://manpages.ubuntu.com/manpages/xenial/en/man8/aa-autodep.8.html
10. **Ivar Jacobson, Grady Booch y James Rumbaugh. 2000.** *El Proceso Unificado de Desarrollo de Software.* [En línea] [Citado el: 4 de Marzo de 2017.] ftp://april.frm.utn.edu.ar/METODOLOGIA%20DE%20SISTEMAS%20-%20TSP/LIBROS/El%20proceso%20unificado%20de%20desarrollo%20de%20software.pdf

11. **Tamara Rodríguez**. **2015**. Metodología de desarrollo para la Actividad productiva de la UCI. La Habana: versión 1.2, 2015.
12. **DesarrolloWeb. 2000.** Qué es Python. [En línea] [Citado el: 22 de Febrero de 2017.] http://desarrolloweb.com/Qué%20es%20Python.htm
13. **Pythoniza. 2014 – 2017.** PyCharm. [En línea] [Citado el: 22 de Febrero de 2017.] https://pythoniza.me/pycharm-community-edition
14. **Ingeniería de Software. 2011.** UML. [En línea] [Citado el: 20 de Enero de 2017.] https://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf
15. **Mike Cohn. 2005**. Agile Estimating and Planing. 2005.
16. **Roger S. Pressman. 2009**. Software Engineering.
17. **Roger S. Pressman.** Ingeniería del Software: Un Enfoque Práctico. Madrid : Mc Graw Hill, 2005.
18. **Universidad de Alicante. 1996 - 2017.** Modelo vista controlador. [En línea] [Citado el: 10 de Marzo de 2017] https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html
19. **Macario Polo Usaola. 2017.** Patrones GRASP. [En línea] [Citado el: 12 de Febrero de 2017.] http://docplayer.es/14133533-Patrones-grasp-macario-polo-usaola-patrones-grasp-1.html

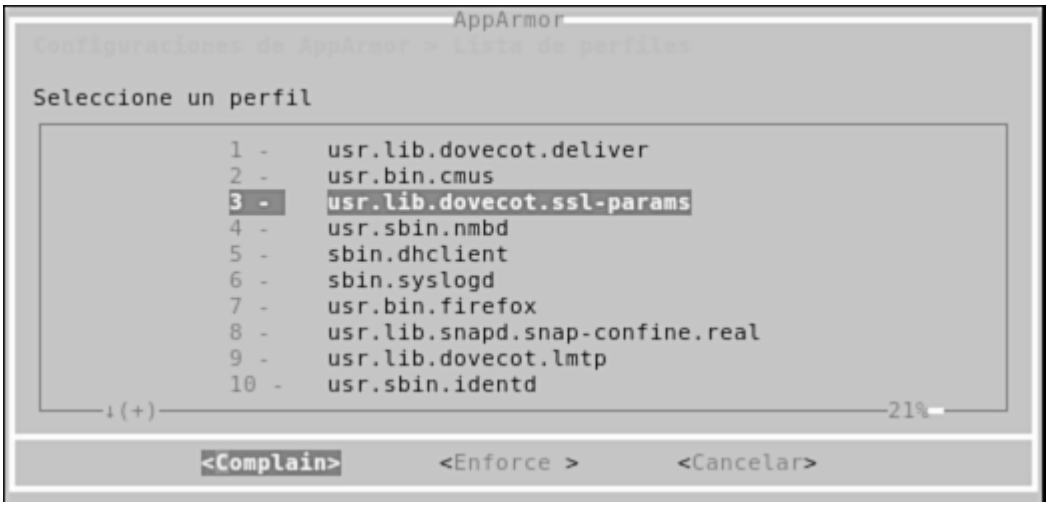
Anexo 1

Historia de Usuario	
Numero: HU_2	**Nombre:** Eliminar perfil
Prioridad de negocio: Alta	**Iteración asignada:** 2
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 1 días
Descripción: Se encarga de eliminar un perfil seleccionado.	
Prototipo: 	

Historia de Usuario	
Numero: HU_7	**Nombre:** Aplicar modo complain
Prioridad de negocio: Alta	**Iteración asignada:** 7
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Se selecciona un perfil para ser monitorizado, y registrar las violaciones de las restricciones, pero sin restringir el acceso.	
Prototipo:	



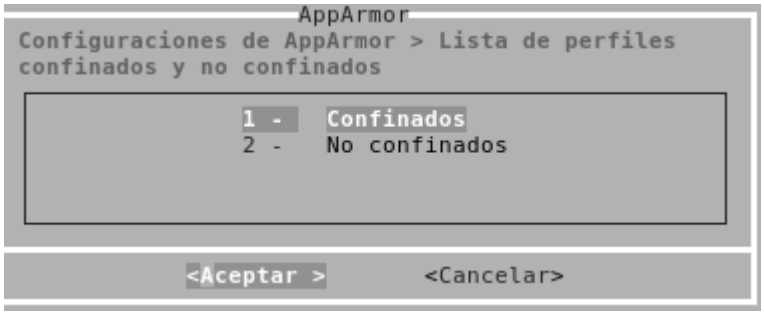
Historia de Usuario	
Numero: HU_5	**Nombre:** Deshabilitar perfil
Prioridad de negocio: Alta	**Iteración asignada:** 5
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Deshabilita el funcionamiento del perfil deseado, sin eliminarlo.	
Prototipo:	

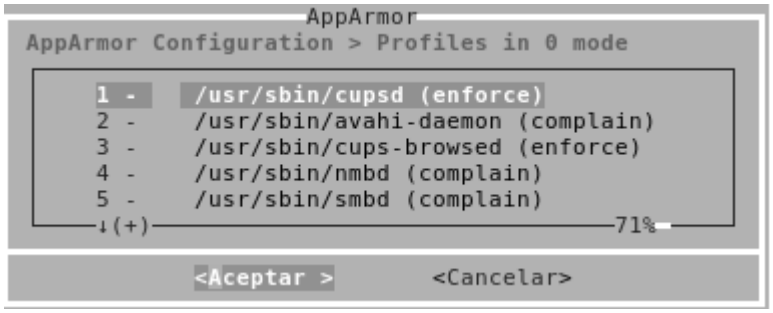
Historia de Usuario	
Numero: HU_8	**Nombre:** Aplicar modo enforce
Prioridad de negocio: Alta	**Iteración asignada:** 8
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Se selecciona un perfil para ser monitorizado, y registrar las violaciones de las restricciones, denegando el acceso al recurso y la aplicación perteneciente al perfil se confina.	
Prototipo:  	

Historia de Usuario	
Numero: HU_9	**Nombre:** Listar aplicaciones con conexiones tcp-udp
Prioridad de negocio: Alta	**Iteración asignada:** 9
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días

Descripción: Lista las aplicaciones con conexiones tcp-udp activas que no están confinadas y las confinadas.

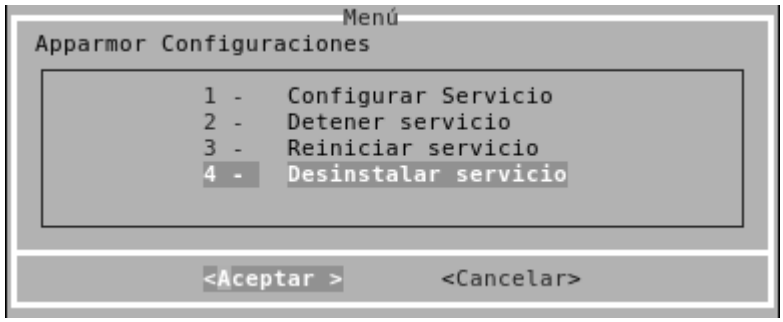
Prototipo:





Historia de Usuario	
Numero: HU_10	**Nombre:** Instalar servicio
Prioridad de negocio: Media	**Iteración asignada:** 10
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Instala el paquete apparmor-utils para garantizar el correcto funcionamiento del módulo	
Prototipo: 	

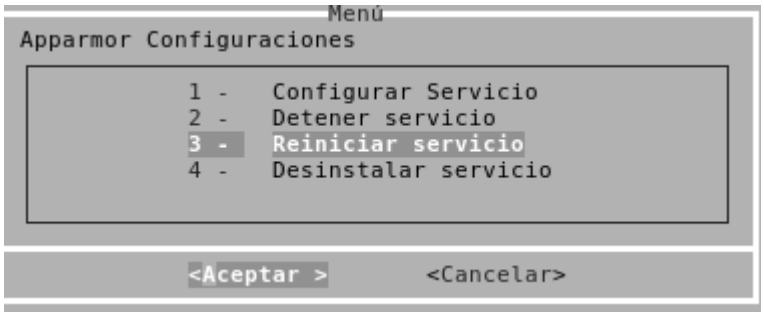
Historia de Usuario	
Numero: HU_11	**Nombre:** Desinstalar servicio
Prioridad de negocio: Baja	**Iteración asignada:** 11
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Desinstala el paquete apparmor-utils	
Prototipo:	



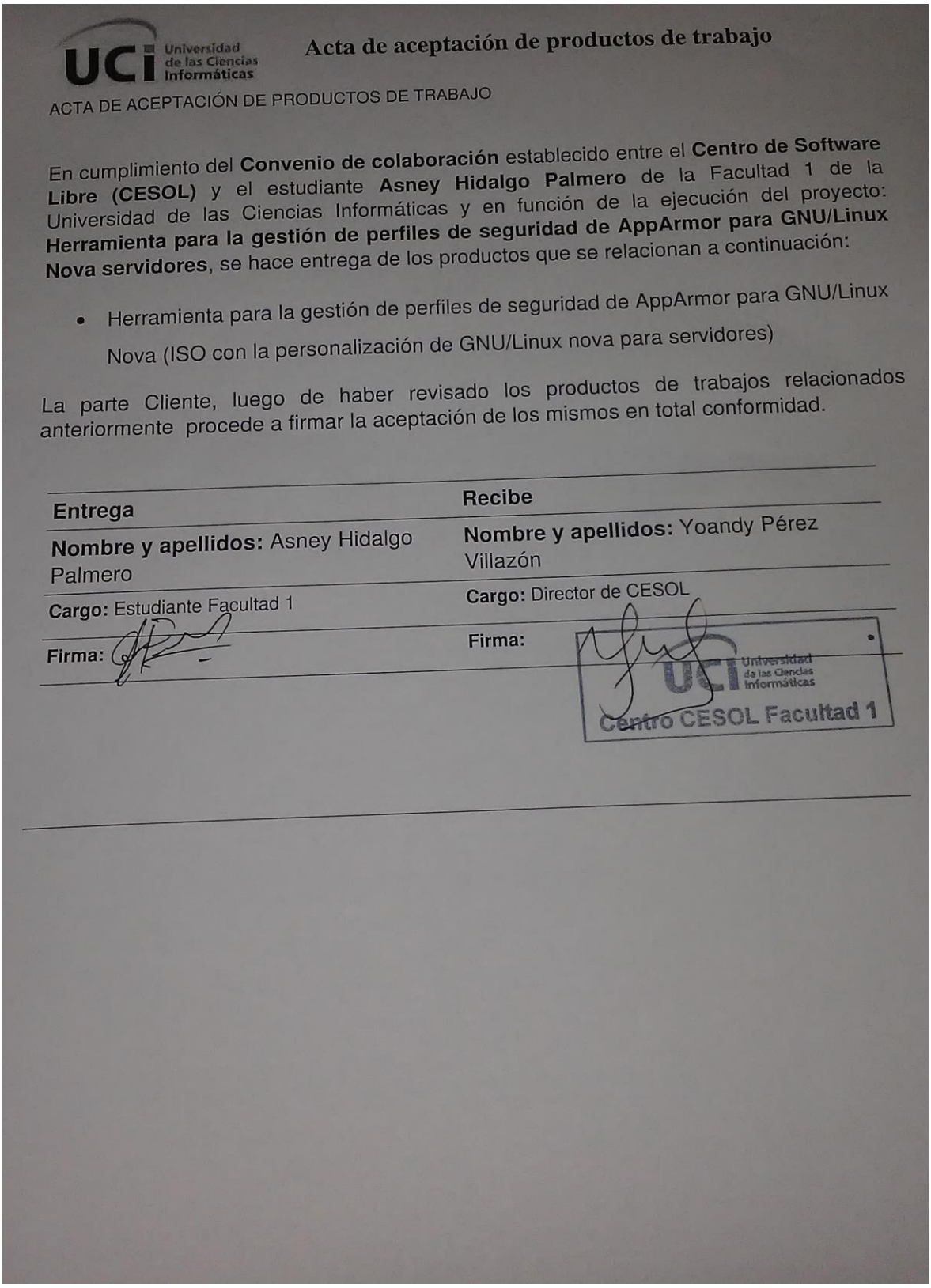
Historia de Usuario	
Numero: HU_12	**Nombre:** Iniciar servicio
Prioridad de negocio: Baja	**Iteración asignada:** 12
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Inicia el servicio de AppArmor	
Prototipo: 	

Historia de Usuario	
Numero: HU_13	**Nombre:** Detener servicio
Prioridad de negocio: Baja	**Iteración asignada:** 13
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Detiene el servicio de AppArmor	
Prototipo: 	

Historia de Usuario	
Numero: HU_14	**Nombre:** Reiniciar servicio
Prioridad de negocio: Baja	**Iteración asignada:** 14
Programador: Asney Hidalgo Palmero	**Tiempo estimado:** 1 días
Riesgo de Desarrollo: -Interrupción del fluido eléctrico	**Tiempo real:** 2 días
Descripción: Detiene el servicio de AppArmor	
Prototipo:	



Anexo 2



UCI Universidad de las Ciencias Informáticas

Acta de aceptación de productos de trabajo

ACTA DE ACEPTACIÓN DE PRODUCTOS DE TRABAJO

En cumplimiento del **Convenio de colaboración** establecido entre el **Centro de Software Libre (CESOL)** y el estudiante **Asney Hidalgo Palmero** de la Facultad 1 de la Universidad de las Ciencias Informáticas y en función de la ejecución del proyecto: **Herramienta para la gestión de perfiles de seguridad de AppArmor para GNU/Linux Nova servidores**, se hace entrega de los productos que se relacionan a continuación:

- Herramienta para la gestión de perfiles de seguridad de AppArmor para GNU/Linux Nova (ISO con la personalización de GNU/Linux nova para servidores)

La parte Cliente, luego de haber revisado los productos de trabajos relacionados anteriormente procede a firmar la aceptación de los mismos en total conformidad.

Entrega	**Recibe**
Nombre y apellidos: Asney Hidalgo Palmero	**Nombre y apellidos:** Yoandy Pérez Villazón
Cargo: Estudiante Facultad 1	**Cargo:** Director de CESOL
Firma:	**Firma:**

UCI Universidad de las Ciencias Informáticas
Centro CESOL Facultad 1