Universidad de las Ciencias Informáticas
Facultad 1

Centralización y visualización de l*ogs* generados por los IDS para Nova Servidores

Trabajo de diploma para optar por el título de ingeniero en Ciencias informáticas

Autor :
Rubén Lage Laricheva

Tutores:
Ms. C. Eylin Hernández Luque
Ing. Joel Ayata Escalona

La Habana, junio 2017

Declaración de autoría

Declaro por este medio que yo Rubén Lage Laricheva con carné de identidad 92042621868 soy el autor principal del trabajo titulado “Centralización y visualización de l*ogs* generados por los IDS para Nova Servidores” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales del mismo con carácter exclusivo. Para que así conste firmo la presente declaración jurada de autoría en La Habana a los _____ días del mes de _____ del año _____.

_____

Rubén Lage Laricheva

_____ _____

Msc. Eylín Hernández Luque Ing. Joel Ayata Escalona

“No enseñar a un hombre que está dispuesto a aprender es desaprovechar a un hombre”

Confucio.

Dedicatoria

A mi familia, que es lo más grande que tengo y tendré, por hacerme sentir como que nunca me falto apoyo en momentos grises, por cuidarme y velar por mí en todos los años de mi vida.

Agradecimientos

- A mi familia, mi papá por siempre saber las palabras exactas con las cual aconsejarme, a mi mamá por quererme tanto y hacerme saber que puedo esperar de ella mucho más de lo que me merezco, mi hermano que ha sabido demostrarme que siempre puedo contar con él y a mi hijo, mi pedazo de cielo, solo su imagen en mi mente me da la energía necesaria para seguir adelante.
- A mis tutores (la profesora Eylín y Ayata sensei), por poner a mi disposición ese basto conocimiento que poseen, por guiarme y hacer más fácil el camino hacia este momento.
- A los amigos que logré hacer en todos estos años en la universidad, los que vienen junto conmigo desde el primer momento: Asney, el lucky, el levy, el cepi, campillo. Los que conocí recientemente: el yerba buena, Raúl, el piquete del apartamento 110103 y el grupo 1502. Otros que ya no están como Julio y Jose.
- A los frikis de mi antiguo pueblo, mis hermanos de Santa Lucía, esos con los que compartí momentos de excesos, tristezas y alegrías.
- A los miembros del tribunal y mi oponente por sus consejos, correcciones y ayuda en estos últimos meses.
- A todos mis profesores a lo largo de la carrera, a pesar de la juventud de la mayoría, fueron todos ejemplos de profesionalismo.
- A mi compañera de la vida y sentimiento, que a pesar de encontrarnos distantes siempre me ha hecho sentir como si estuviéramos uno al lado del otro y su cariño y afecto nunca me ha faltado.

Resumen

Como resultado del proceso de migración de *software* que ha llevado a cabo Cuba se ha desplegado en varias instituciones del país el sistema operativo Nova, desarrollado en el Centro de Software Libre de la Universidad de las Ciencias Informáticas. Por motivos de seguridad se necesita la utilización de un Sistema de Detección de Intrusos para este sistema operativo en su versión para servidores. Esta herramienta necesita ser monitoreada constantemente, proceso que es extremadamente engorroso debido a la cantidad de registros (*logs*) que genera de forma local. Para resolver este problema la presente investigación propone: desarrollar un sistema para la centralización y visualización de dichos registros. Para esto se fundamentan teóricamente las herramientas que realizan el proceso de centralización de eventos, que permitirá conocer el estado de las mismas, se realiza un proceso de evaluación y selección de los Sistemas de Detección de Intrusos existentes y de las tecnologías adecuadas para realizar la visualización, utilizando el método de Calificación y Selección de *Softwares* de Código Abierto (QSOS). Se define como metodología de desarrollo AUP en su variante para la UCI, la cual guiará el proceso de desarrollo de software. El resultado esperado de la investigación contribuye un sistema capaz de centralizar y visualizar los *logs* del IDS en Nova Servidores, lo cual permitirá la revisión de todos estos en un mismo ordenador y en tiempo real.

Palabras clave: Nova Servidores, IDS, *logs*, centralización, visualización.

Sumario

Índice de ilustraciones

Índice de tablas

Centralización y Visualización de *logs* del IDS para Nova Servidores

Introducción

En el mundo de las Tecnologías de la Información y las Comunicaciones (TIC's) el *software* representa un componente fundamental, el cual se puede clasificar en privativo o libre atendiendo a las libertades de los usuarios para trabajar con ellos. El *software* libre es aquel que le permite al usuario su copia, ejecución, mejora, distribución y modificación. Debido a estas libertades que proporciona este tipo de *software* muchas instituciones en el mundo han llevado a cabo un proceso de sustitución de programas privativos por programas libres, lo que se conoce como proceso de migración a *software* libre, buscando perfeccionar aspectos en sus sistemas informáticos entre los cuales uno de los más supervisados es la seguridad, ya que necesita una mejora continua y constituye uno de los factores más vulnerables.

Con el crecimiento que ha tenido la utilización de las redes y los servicios teleinformáticos, la seguridad ha constituido un aspecto indispensable para garantizar la confidencialidad, integridad y disponibilidad de la información. Actualmente, los ataques por *hackers* a los servidores de las grandes compañías del mundo y de gobiernos se han incrementado. Según el periódico colombiano "El espectador", se registraron 7 118 ciberataques en 2015 y entre 2014 y ese mismo año hubo un incremento del 40% de estos ataques [1]. A esto se suma el desarrollo de herramientas informáticas que permiten encontrar vulnerabilidades en redes con facilidad a personas que no necesariamente deban tener un conocimiento amplio de la informática, lo que significa que nadie está a salvo de ellos.

Cuba lleva a cabo un proceso migratorio a *software* libre impulsada por motivos como la independencia tecnológica, el factor económico relativo al pago de licencias y las dificultades para adquirir el software privativo [2]. En la Universidad de las Ciencias Informáticas (UCI), se creó la distribución GNU/Linux Nova[1] con el objetivo de alcanzar el desarrollo de un sistema operativo altamente confiable y que cumpla con las necesidades actuales del país. Entre los aspectos en los que está enfocado en alcanzar niveles de excelencia este sistema operativo, se encuentra la seguridad [3]. Un paso de avance hacia este planteamiento sería el despliegue de un IDS para esta distribución en su versión para servidores. El tráfico de las redes de las instituciones que utilizarán Nova Servidores posibilitará que el IDS constantemente detecte eventos que considere como amenazas, generando una cantidad de información elevada. Aunque

[1] Distribución cubana de GNU/Linux

está información posibilitará estar al tanto de lo que ocurre en la red, no será posible analizarla fácilmente.

Un IDS se utiliza para monitorizar el tráfico de la red en un determinado sistema informático con el objetivo de detectar intentos que comprometan la seguridad de dicho sistema. Este es capaz de generar *logs* de paquetes IP[2] y se puede usar para detectar gran variedad de ataques o intentos, tales como *buffer overflows*[3] y escaneo de puertos silenciosos[4] [4].

Estas características convierten al IDS en una herramienta muy confiable a la hora de monitorizar un sistema de redes. Pero, a la vez que brinda muchas ventajas en la seguridad, posee también desventajas a la hora de su utilización. Con el uso de métodos científicos tales como el análisis documental, entrevistas a especialistas en el área del conocimiento y experimentos hechos por el autor de este trabajo, se comprobó que estas herramientas son capaces de generar aproximadamente 50 alertas por hora y sus *logs*[5] pueden llegar a contener cerca de 500 alertas monitoreando una sola subred.

Entonces, si se cuenta con una red con un mayor rango de direcciones y que tenga varios IDS instalados, no sería erróneo mencionar que la revisión de todos esos *logs* que generarían dichos sistemas sería un proceso engorroso y difícil. Lo cual se debe a que estas herramientas en la actualidad, son incapaces de centralizar y visualizar la información que generan.

La gestión local de *logs* del IDS obliga al administrador de red a acceder a cada IDS instalado en los servidores y revisar los *logs* generados por estos en la carpeta var/log/IDS , lo que le consumirá una gran cantidad de tiempo. Este consumo de tiempo se multiplicará si se despliegan más IDS en la red, además no será posible le revisión de dichos *logs* en tiempo real y la visibilidad de los mismos será compleja. La búsqueda de eventos específicos se dificulta ya que no existen opciones para filtrar por los campos que contienen las alertas (fecha, dirección IP, ataque).

Partiendo de esta situación se plantea el siguiente **problema de investigación:** ¿Cómo lograr la

[2] Internet Protocol.

[3] Un *buffer overflow* (o desborde de memoria) se lleva a cabo cuando un programa informático excede el uso de cantidad de memoria asignado por el sistema operativo.

[4] Acción de analizar por medio de un programa el estado de los puertos de una máquina conectada a una red de comunicaciones. Detecta si un puerto está abierto, cerrado, o protegido por un cortafuego.

[5] Fichero que guarda información acerca del sistema que lo genera.

centralización y visualización de los *logs* generados por el IDS de Nova servidores y facilitar la revisión de los mismos?

Se trazó como **objeto de estudio** el proceso de centralización y visualización de *logs* en los sistemas GNU/Linux.

Se define como **campo de acción** la centralización y visualización de *logs* en la distribución Nova servidores y como **objetivo general,** desarrollar un sistema, seleccionando las herramientas y tecnologías adecuadas, que permita la centralización y visualización de los *logs* generados por el IDS en Nova Servidores y que facilite el proceso de monitoreo en la red.

Para cumplir con el objetivo general planteado se establecen los siguientes **objetivos específicos:**

- Realizar el marco teórico referente a las herramientas de centralización y visualización de *logs* existentes para distribuciones de GNU/Linux.
- Sistematizar acerca del IDS en los sistemas GNU/Linux.
- Diseñar un sistema para la centralización y visualización de *logs* del IDS en Nova servidores.
- Implementar las funcionalidades del sistema para la centralización y visualización de *logs* del IDS en Nova servidores.
- Realizar pruebas funcionales al sistema para la centralización y visualización de *logs* del IDS en Nova servidores.

Para orientar la investigación, se proponen las siguientes **tareas de investigación**:

- Sistematización de los referentes teóricos sobre los IDS, las herramientas de centralización y visualización de *logs* en los sistemas GNU/Linux y modelos de madurez para la evaluación y elección de *softwares* de código abierto.
- Selección de las tecnologías a utilizar para el desarrollo del sistema que permita la centralización y visualización de los *logs* generados por los IDS en Nova servidores.

- Modelación de los artefactos que se generan con la metodología seleccionada.
- Codificación de las funcionalidades definidas.
- Valoración de la propuesta.

El presupuesto hipotético de la solución se centra en las siguientes **preguntas científicas:**

1. ¿Cuáles son los referentes teóricos que sustentan el desarrollo de la investigación, relacionados con la centralización y visualización de *logs* del IDS para Nova servidores?
2. ¿Cuáles son las características y componentes de las herramientas, tecnologías y metodologías existentes que permiten la construcción de la centralización y visualización de los *logs* del IDS para Nova Servidores?
3. ¿Qué estructura tienen los elementos que deben tenerse en cuenta para realizar el análisis y diseño de la centralización y visualización de *logs* del IDS para Nova servidores?
4. ¿Cómo contribuye la propuesta a la centralización y visualización de *logs* del IDS para Nova Servidores para que facilite el proceso de monitoreo en la red?

El **resultado esperado** es obtener un sistema que centralice y visualice los *logs* generados por el IDS en Nova servidores.

Se hará uso de diferentes **métodos científicos** para realizar la investigación. **Los métodos teóricos** que se utilizan son el **analítico-sintético**, **inductivo-deductivo** y **modelación.** El método analítico-sintético se utiliza para buscar y analizar documentos e información sobre las herramientas de centralización de *logs* e IDS. Por otra parte, el método inductivo-deductivo permite llegar a proposiciones generales a partir de los hechos que conforman la teoría, o a partir de esa teoría arribar a conclusiones sobre casos particulares que se llevaron a la práctica. En la investigación se utiliza este método para seleccionar las características que va a poseer el sistema a desarrollar en este trabajo de forma tal que realice el proceso de centralización y visualización de los *logs* del IDS en Nova Servidores correctamente. El método modelación es utilizado para confeccionar los diagramas correspondientes a la representación de la

propuesta de solución.

Los **métodos empíricos** que se utilizarán serán el **análisis documental, la entrevista** y **el experimento.** El análisis documental es empleado para la revisión de literatura durante la investigación sobre el proceso de centralización y visualización de *logs*. El método de entrevista se utiliza en la búsqueda de apoyo en el personal especializado en el tema y el experimento se utilizó para obtener resultados que demuestren la existencia de una situación problemática.

El presente documento se encuentra estructurado de la forma siguiente:

Capítulo 1. Fundamentación teórica. En este capítulo se hace un análisis de las herramientas existentes para la centralización de información, se analizan también las tecnologías a utilizar para realizar el proceso de centralización y visualización de *logs* y se definen varios conceptos relacionados con el tema, así como la metodología a utilizar.

Capitulo 2. Análisis y diseño de la solución. Aquí se realiza la extracción de los requisitos funcionales y no funcionales de la aplicación a desarrollar. A través de los diagramas de clase y de paquetes, se modelan las reglas de negocio, la estructura y funcionamiento de la solución. Además, se seleccionan los patrones de diseño que serán usados para resolver la problemática.

Capítulo 3. Implementación y prueba de la solución. Se realiza el análisis de las funciones identificadas como vulnerables, se muestra el diagrama de componentes, así como las pruebas realizadas al sistema y la validación de la solución.

Capítulo 1. Fundamentación Teórica

Introducción

En el presente capítulo se definen conceptos asociados al tema investigativo. También, se realiza un análisis de las herramientas que existen y se utilizan en la centralización y visualización de información, se lleva a cabo un estudio que incluye las tecnologías actuales para el desarrollo de software y las principales herramientas y tecnologías que serán utilizadas para la implementación de la solución.

1.1 Definiciones asociadas al tema investigativo

Log

Un *log* es un fichero que puede registrar mucha información acerca de eventos relacionados con el sistema que la genera. Estos ficheros contienen información muy útil para la recuperación de datos, permiten detectar incidentes de seguridad, son de gran ayuda como evidencia legal y constituyen el punto de partida del análisis forense [5].

Los *logs* representan una de las mayores fuentes de información cuando se gestionan incidentes de seguridad. Tanto a nivel de comunicaciones (routers, cortafuegos, IDS) como a nivel de sistema (Unix, Windows, Linux) y de aplicación (servidor web, servidor de correo). Desde que se inicia una sesión hasta que se cierra en una computadora se almacenan todas las acciones realizadas mientras la sesión estuvo abierta. La información que almacenan los *logs* se puede utilizar como pruebas legales de cualquier violación que se haya cometido en cierta red, por lo que puede ser utilizada como evidencia digital ante la ley [6].

Centralización de *logs*

La centralización se refiere a reunir varias cosas en un centro común o hacer que distintas cosas dependan de un poder central [7]. Por lo que se puede definir como centralización de *logs* al agrupamiento de estos en una herramienta de almacenamiento común, como una base de datos.

Importancia de centralizar los *logs*

Los *logs* generados por las aplicaciones representan la información acerca de los eventos ocurridos en sus períodos de ejecución. Al almacenarse en sus propios ficheros causan problemas de visibilidad al

usuario. La forma de evitar esto es centralizándolos. La agrupación de los ficheros de *logs* en un único sitio facilita analizar el rendimiento de ese sistema y plantear continuamente posibles mejoras [8].

Ventajas de la centralización de *logs*

La centralización de *logs* permite la detección de errores, depurar las aplicaciones, o sea, realizar las pruebas necesarias para prevenir posibles problemas. También hace posible la gestión de excepciones, configuración de alarmas y disparadores y la extracción de información adicional, lo que quiere decir que se puede acceder a datos secundarios, como por ejemplo a los códigos HTTP[6] de solicitudes a servidores [8].

Intrusión

Una intrusión es un conjunto de acciones que intentan comprometer (poner en peligro) la integridad, la confidencialidad o la disponibilidad de un sistema informático. Las intrusiones tienen distintos orígenes: atacantes que acceden a los sistemas desde Internet, usuarios autorizados del sistema que intentan ganar privilegios adicionales para los cuales no están autorizados, usuarios autorizados que hacen un mal uso de los privilegios o recursos que se les han asignado.

De una forma o de otra las intrusiones están dirigidas fundamentalmente a:

- Acceder a una determinada información.
- Manipular cierta información.
- Hacer que el sistema no funcione de forma segura o inutilizarlo [8].

IDS

Un IDS es una herramienta que intenta detectar o monitorizar los eventos ocurridos en un determinado sistema informático o red informática en busca de intentos que constituyan amenazas para dicha red [1]. Estas herramientas se clasifican en tres departamentos: fuentes de información, tipo de análisis y tipo de respuesta. Según las fuentes de información los IDS se clasifican en IDS basados en red (NIDS) e IDS basados en host (HIDS), en cuanto al tipo de análisis se clasifican en detección de mal uso y detección de

[6] Es un protocolo de transferencia de información entre diferentes servicios y clientes que utilizan páginas web.

uso anómalo y según el tipo de respuestas se clasifican en activos o pasivos [3].

NIDS

Un NIDS revisa los paquetes que transitan en la red en busca de elementos que denoten un ataque en contra de algunos de los sistemas ubicados en ella. Estos tipos de IDS pueden situarse en cualquiera de las computadoras o en un elemento que analice todo el tráfico, como un repetidor. Esté donde esté, protegerá varias computadoras y no una sola [3].

Localización para los NIDS

Existen diferentes localizaciones en la red donde un IDS puede ubicarse y según estas localizaciones se pueden definir acciones que estas herramientas puedan realizar:

- Delante del cortafuego externo
- Detrás del cortafuego externo
- En las redes principales
- En las subredes de valor crítico
- En los host o pc clientes

Delante del cortafuego externo

Colocar los agentes delante del cortafuego externo permite monitorizar el número y tipo de ataques dirigidos contra la infraestructura de la organización y detectar ataques cuyo objetivo es el cortafuego principal. Esta posición también presenta algunos inconvenientes, por ejemplo, no permite detectar ataques que utilicen en sus comunicaciones algún método para ocultar la información, como algoritmos de encriptación, el NIDS puede convertirse en un blanco fácil si algún atacante logra identificarlo [3].

Detrás del cortafuego externo

Esta localización situada entre internet y la red interna se denomina Zona Desmilitarizada (DMZ) y permite

monitorizar intrusiones que logran atravesar el cortafuego principal, además se pueden detectar ataques dirigidos contra los servidores que ofrecen servicios públicos situados en esta subred. Aunque también existen limitaciones en esta localización pues, aunque la seguridad del NIDS mejora con la inclusión del cortafuego que lo separa de la red exterior, esto no excluye tomar medidas adicionales para evitar que pueda ser comprometido por atacantes [3].

En las redes principales

Cuando se monitoriza el tráfico de red en las redes con mayor actividad se obtiene más cantidad de tráfico por lo tanto hay también más posibilidades de encontrar posibles ataques, además se pueden detectar ataques producidos desde dentro de la propia red, como los realizados por personal interno, aunque desde esta posición tampoco permite detectar ataques que utilicen algoritmos de encriptación en sus comunicaciones [3].

Subredes de valor crítico

Desde esta posición el NIDS puede detectar ataques realizados contra elementos críticos de la red y dedicar especial atención a los recursos más valiosos de la infraestructura, aunque no evitan problemas de monitorización relacionados con el uso de conmutadores y no están estratégicamente bien situados ante ataques de origen interno [3].

En Host o pc clientes

En esta posición el NIDS es capaz de evitar los inconvenientes de la encriptación de las comunicaciones presentes en las localizaciones anteriores, pero la visión del sistema de detección está limitada tanto por la situación de la máquina, como por la arquitectura de la red, por ejemplo, si se utilizan conmutadores, sólo puede analizar el tráfico relacionado con la máquina anfitriona [3].

HIDS

Este tipo de sistemas necesita ser configurado de forma individual en cada máquina y utiliza como fuente de datos la información obtenida del sistema. Se utilizan para acceder a la información recolectada por las herramientas de auditorías del host (registro de actividad, accesos al sistema de ficheros y *logs* de

registro) [3]. Estos sistemas actúan monitorizando cambios del sistema a nivel local por tanto llegan a un nivel más bajo que otros sistemas de detección de intrusos y pueden detectar intentos de intrusión que otros sistemas no pueden detectar. Sin embargo estos sistemas actúan a nivel de host y requieren más tiempo de gestión y configuración, ya que cada host donde lo implantemos deberá tener una configuración específica [9].

Requisitos de los IDS

Un IDS debe poseer ejecución autónoma continua, o sea, debe ejecutarse continuamente sin nadie que esté obligado a supervisarlo. Aunque al detectar un problema se informa al administrador o se lanza una respuesta automática, el funcionamiento habitual no tiene que implicar interacción con un humano. No debe introducir cambios en el comportamiento habitual del sistema que protege, lo que quiere decir que no han de introducir una sobrecarga en el sistema, ni generar una cantidad elevada de falsos positivos de logs. También un IDS debe ser tolerante a fallos, lo que quiere decir que debe ser capaz de responder ante situaciones inesperadas. Algunos de los cambios que se pueden producir en un entorno informático no son graduales sino bruscos y el IDS debe ser capaz de responder siempre adecuadamente ante los mismos [3].

1.2 Modelo de madurez

Para lograr realizar una exitosa evaluación y selección de las herramientas de código abierto candidatas a ser utilizadas en la solución es necesario la realización de un modelo de madurez enfocado en herramientas para software libre. Existen dos formas de realizar este proceso: la que establece el Modelo de Madurez de Código Abierto (OSMM) y la que establece la Calificación y Selección de Software de Código Abierto (QSOS). En el trabajo se utilizará la segunda opción ya que es la que se utiliza en el Centro de Software Libre (CESOL) de la UCI.

1.2.1 QSOS

QSOS es un proyecto libre con el objetivo de mutualizar y capitalizar la vigilancia tecnológica en componentes y proyectos de código abierto. Está compuesto por un método formal que describe un flujo

de trabajo para evaluar estos componentes y proyectos, una serie de herramientas para ayudar a aplicar el flujo de trabajo en plantillas, evaluaciones y comparaciones y una comunidad desarrollando esas plantillas, evaluaciones y herramientas [10].

Método

El método de QSOS establece 4 pasos: “definición” es el paso donde se constituye y enriquecen los marcos de referencia usados en los siguientes pasos, “evaluación” es el encargado de evaluar el software de acuerdo a tres ejes de criterios (cobertura funcional, riesgos para el usuario y riesgos para el proveedor de servicio), el paso “calificación” es el que establece la ponderación de los criterios divididos en los tres ejes mencionado anteriormente, modelando el contexto y “selección” aplica el filtro establecido en el paso anterior. Basándose en estos pasos y sus criterios este método lleva a cabo la puntuación. Los criterios son puntuados del 0 al 2 [11].

1.3 Herramientas para la visualización de *logs*.

Kibana

Kibana es una plataforma de visualización de datos con licencia Apache 2.0 de código abierto que ayuda en la visualización de datos estructurados y no estructurados almacenados en los índices de *Elasticsearch*. Kibana está enteramente escrito en *HTML* y *JavaScript*. Utiliza las capacidades de búsqueda e indexación de *Elasticsearch* expuestas a través de su *API RESTful* para mostrar gráficos poderosos para los usuarios finales. Desde la inteligencia de negocio básica hasta la depuración en tiempo real, desempeña su papel a través de la exposición de datos a través de histogramas, geomapas, gráficos circulares y tablas.

Facilita la comprensión de grandes volúmenes de datos. Su sencillo navegador basado en Interfaz le permite crear y compartir rápidamente *dashboards* dinámicos que muestran cambios en las consultas de *Elasticsearch* en tiempo real.

Algunas de las características claves de *Kibana* son las siguientes:

- Ofrece un análisis flexible y una plataforma de visualización para inteligencia.

- Proporciona análisis en tiempo real, resumen, creación de gráficos y capacidades de depuración
- Proporciona una interfaz intuitiva y fácil de usar, que es altamente personalizable mediante algunas funciones de arrastrar y soltar y alineaciones cuando sea necesario.
- Los paneles pueden ser fácilmente compartidos e incrustados en diferentes sistemas.
- Permite compartir instantáneas de registro que ya ha buscado.
- Permite guardar los *dashboards* y gestionar más de uno, los cuales pueden ser compartidos e incrustados en diferentes sistemas [12].

Grafana

Grafana es una plataforma de visualización y análisis de métrica de código abierto. Es comúnmente usada para visualizar datos de series de tiempo para análisis de infraestructura y aplicaciones, pero muchos la usan en otros dominios incluyendo sensores industriales, tiempo y control de procesos [13]. Soporta muchos *backends* de almacenamiento diferentes para sus datos de series de tiempo (*Data Source*). Cada origen de datos tiene un editor de consultas específico que se personaliza para las características y capacidades que el origen de datos específico expone. También es compatible con múltiples organizaciones con el fin de soportar una amplia variedad de modelos de implementación, incluyendo el uso de una sola instancia de Grafana para proporcionar servicio a varias organizaciones potencialmente no confiables. Otro aspecto a tomar en cuenta es que brinda una amplia variedad de estilos y formatos para crear imágenes mediante sus paneles y como funcionalidad principal permite la creación de *dashboards* los cuales pueden ser definidos por un conjunto de uno o más paneles organizados y acomodados en una o más filas. Al igual que en Kibana, el período de tiempo del *Dashboard* puede ser controlado por el “time picker” en la esquina superior derecha, estos pueden utilizar plantillas para ser más dinámicos e interactivos y anotaciones para desplegar datos a través del panel [14].

Elección de la herramienta para la visualización

Habiéndose analizado las dos herramientas candidatas para realizar el proceso de visualización se lleva a cabo la evaluación de ambas para la posterior elección de la más madura de ellas de acuerdo con el método para la Calificación y Selección de Software de Código Abierto (QSOS) descrito anteriormente. Se

debe tener en cuenta que se evaluarán las características de interés para la solución:

Edad: Mientras más antiguo es el software, poseerá mayor madurez.

Estabilidad: Cuan estable es un software en cuanto a sus lanzamientos de nuevas versiones e implementación de nuevas funcionalidades.

Administración/Supervisión: Existencias de funcionalidades para administrar o supervisar el rendimiento.

Documentación: el grado de disponibilidad de la documentación debe ser tenido en cuenta. La existencia de foros de discusión, listas de correo, manuales de uso y libros son elementos a considerar.

Facilidad de uso: permite analizar la facilidad de utilizar un mecanismo. Se debe tener en cuenta el más fácil de utilizar.

Las características funcionales serán evaluadas con una puntuación del 0 al dos. Siendo las funciones de puntuación 2 las mejores. Las principales funcionalidades que se tomarán en cuenta en el trabajo con las que debe cumplir una herramienta de visualización son:

- Permitir la construcción de diferentes tipos de gráficos.
- Salvar y compartir las visualizaciones construidas.
- Permitir especificar los campos que se desean visualizar.
- Permitir la búsqueda y exploración de datos.
- Enfoque en análisis de *logs*.

Evaluación entre las herramientas.

Generales	**Grafana**	**Kibana**
Edad	3 años	4 años
Estabilidad	Estable	Estable

Administración/Supervisión	Sin funcionalidades de supervisión o administración definidas.	Kibana cuenta con un paquete de funcionalidades que contiene la opción de mantener un pulso en su rendimiento y en el de *Elasticsearch* y *Logstash*.
Documentación	Si	Si
Facilidad de uso	El soporte de muchos *backends* para almacenar sus datos hace que su instalación y uso sea más complejo.	Su unión con el *stack* ELK hace que sea fácil de instalar y de usar.
Funcionales		
Construcción de diferentes tipos de gráficos	2	2
Salvar y compartir	2	2
Búsqueda y exploración de datos	0	2
Enfoque en análisis de *logs*	1	2

Tabla 1: Evaluación entre Kibana y Grafana.

Al analizar la comparación anterior se observa como Kibana supera en ambos tipos de características a Grafana. Por lo que se propone como resultado de la evaluación, seleccionar a Kibana como herramienta de visualización para la solución.

ELK

AL escogerse Kibana como herramienta de visualización, es imperativa la utilización del *stack* ELK en el trabajo, por lo que además de Kibana se usará *Elasticsearch* y *logstash.* Esto hace necesario un estudio de estas dos herramientas.

La plataforma ELK es una completa solución de análisis de *logs*, construida con la combinación de tres herramientas de código abierto: ***Elasticsearch*, *Logtash*, y *Kibana*.** ELK utiliza el *stack* de código abierto de *Elasticsearch* para búsquedas profundas y análisis de datos, *Logtash* para la gestión de *logs* centralizados, lo que incluye el envío y reenvío de *logs* desde servidores múltiples, enriquecimiento de

logs y parseo; y finalmente *Kibana* para visualizaciones poderosas de datos [12].

Elasticsearch

Es un motor de búsqueda distribuido de código abierto basado en *Apache Lucene*[7], y liberado bajo la licencia de Apache 2.0. Proporciona escalabilidad horizontal, confiabilidad y capacidad multientrante para la búsqueda en tiempo real. Las funcionalidades de *Elasticsearch* están disponibles a través de *JSON*[8] sobre una *API RESTful*[9]. Las capacidades de búsqueda están respaldadas por un *Apache Lucene Engine*, que le permite indexar dinámicamente los datos sin conocer la estructura de antemano. *Elasticsearch* es capaz de alcanzar respuestas de búsquedas rápidas porque utiliza la indexación para buscar en los textos [12].

Algunas de las principales funcionalidades de *Elasticsearch* son:

- Se trata de una tienda de documentos distribuida de código abierto escalable y altamente disponible en tiempo real.
- Proporciona capacidades de búsqueda y análisis en tiempo real
- Proporciona una sofisticada *API RESTful* para jugar con la búsqueda y varias funciones, como la búsqueda multilingüe, geolocalización, autocompletar, sugerencias contextuales de "*did-u-mean"*, y fragmentos de resultados [12].

Logtash

Logstash es una tubería de datos que ayuda a recopilar, parsear y analizar una gran variedad de datos estructurados y no estructurados y eventos generados a través de varios sistemas. Proporciona

[7] Apache Lucene es una biblioteca de motor de búsqueda de texto de alto rendimiento y completo, escrita completamente en Java. Es una tecnología adecuada para prácticamente cualquier aplicación que requiera la búsqueda de texto completo, especialmente la plataforma multiplataforma.

[8] Estructura que se utiliza para intercambio de información, presenta un formato {llave: valor}.

[9] Una API RESTful es una interfaz de programa de aplicación (API) que utiliza solicitudes HTTP para realizar acciones de GET, POST, and DELETE datos.

complementos para conectarse a varios tipos de fuentes de entradas y plataformas, y está diseñado para procesar eficientemente registros, eventos y fuentes de datos no estructuradas para la distribución en una variedad de salidas con el uso de sus *plugins* de salida, archivos, *Stdout* (como salida en la consola que ejecuta *Logstash*) o *Elasticsearch.*

Posee las siguientes funcionalidades:

Procesamiento centralizado de datos: *Logstash* ayuda a construir una línea de datos que puede centralizar el procesamiento de datos. Con el uso de una variedad de *plugins* para salidas, puede convertir muchas fuentes de entrada diferentes a un solo formato común.

Soporte para formatos de registro personalizados: los registros escritos por diferentes aplicaciones a menudo tienen formatos específicos para la aplicación. *Logstash* ayuda a analizar y procesar formatos personalizados a gran escala. Proporciona apoyo para escribir tus propios filtros para la tokenización y también proporciona filtros listos para usar.

Desarrollo de *plugins*[10]: Los *plugins* personalizados pueden ser desarrollados y publicados, y existe una gran variedad de complementos desarrollados a medida ya disponibles [12].

1.4 Sistemas de Detección de Intrusos

La selección del IDS es fundamental en este trabajo. Es por eso que se realiza un estudio de los principales IDS existentes y así poder elegir el más adecuado para la solución.

Alienvault[11] y *Upguard*[12], organizaciones de referencia mundial en cuanto a la seguridad de la información establecieron un *ranking* de los IDS para sistemas de código abierto más usados en la actualidad, coincidiendo las dos en que **Snort y Suricata** son los dos IDS más usados por la comunidad de software libre [15]. Debido a esto se decidió analizar estás dos herramientas para escoger la más adecuada para la

[10] Es una aplicación que se relaciona con otra para aportarle una funcionalidad nueva y generalmente muy específica.

[11] Es una compañía desarrolladora de soluciones comerciales y de código abierto para gestionar ataques cibernéticos, incluyendo la plataforma OSSIM.

[12] Es una compañía de arranque de resiliencia cibernética que determina el riesgo cibernético de una compañía escaneando su red interna y sistemas.

solución.

Snort

Snort es un Sistema de detección de intrusiones basado en red (NIDS)[13]. Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía previamente definida como patrones que corresponden a ataques, barridos, intentos de aprovechar alguna vulnerabilidad y análisis de protocolos, por lo que se incluye en la categoría de IDS/IPS[14]. Todo esto en tiempo real [16].

Snort es el IDS de código abierto más antiguo y más probado, además tiene amplia documentación para muchas distribuciones de Linux y posee una cantidad de usuarios cercana a 400 000 personas [17].

La arquitectura de Snort es un eficiente motor de análisis de un solo hilo. La adquisición de paquetes es el primer paso del análisis de tráfico; Snort soporta una amplia variedad de motores de captura de paquetes. Su arquitectura básica utiliza la librería de captura de paquetes Libcap. Sus versiones más recientes poseen un rendimiento de 200-500Mbits/seg de tráfico [18].

Un dato importante a tomar en cuenta en el análisis de estas herramientas es el consumo de recursos computacionales como uso de CPU y RAM. Experimentos realizados por la Escuela Naval de Postgrado de Monterrey, California, aseguran que Snort usa entre un 60% y 70 % para un CPU y su uso máximo de memoria no sobrepasa 1.0 Gbytes [19].

Suricata

Suricata es un motor de detección de amenazas de red, maduro, rápido y robusto, de código abierto y gratuito. Es capaz de detectar intrusiones en tiempo real (IDS), prevenirlas en línea (IPS) y supervisar la seguridad de red. Inspecciona el tráfico utilizando un lenguaje de firmas y cuenta con un soporte de secuencias de comandos [20].

Suricata al igual que Snort soporta una amplia variedad de métodos de adquisición de paquetes, pero a diferencia de este su arquitectura constituye un motor multihilos lo que le permite al motor tomar ventajas de las arquitecturas múltiple núcleo y multiprocesador de los sistemas actuales, además incorpora nuevas

[13] *Network Intrusion Detection System* , lo que se traduce al español como Sistema de Detección de Intrusos de tipo Red

[14] Además de detectar intrusiones también es capaz de prevenirlas y responder ante ellas.

funcionalidades como *ip reputation*[15] . Mediante pruebas oficiales a esta herramienta se pudo comprobar que se ejecuta con un rendimiento de aproximadamente 200Mbits/sec con reglas cargadas. Las últimas versiones de Suricata mejoraron significativamente su rendimiento por núcleo. Una tarjeta de interfaz de red puede producir una mejora significativa del rendimiento al proporcionar varias colas de memoria intermedia (buffer) a los CPU, lo que aumenta el rendimiento de multihilos [18].

En cuanto al consumo de recursos computacionales se demostró en el mismo experimento de la Escuela Naval de Postgrado citado anteriormente que Suricata trabajando con 4 CPU mantiene un promedio de entre 50% y 60% de uso para cada uno de ellos pero usa cada CPU individual a una velocidad menor que la de Snort . En cuanto al uso de RAM estuvo cerca de 3.3 Gbytes [19].

Selección del IDS

Después de analizar estas dos herramientas es necesario evaluarlas y seleccionar una de ellas para utilizarla en la solución utilizando el método de QSOS. Las características funcionales a tomar en cuenta en este trabajo para un IDS son:

- Prevenir ataques (IPS)
- Alertar
- Información de *logs*
- Barrido de puertos

Generales	Snort	Suricata
Edad	19 años	8 años
Popularidad	Aproximadamente 400 000 usuarios	Menor cantidad de usuarios
Soporte	Los proveedores del servicio están fuertemente comprometidos.	Los proveedores del servicio están fuertemente comprometidos.
Documentación	Amplia documentación	Amplia Documentación, pero menor que la de Snort debido a la diferencia de edad.
Facilidad de uso	Fácil	Debido también a que es más joven que Snort, posee menor cantidad de opciones de

[15] Suricata puede señalar el tráfico de fuentes peligrosas conocidas.

		configuración para su uso.
Funcionales		
Prevenir Ataques	2	2
Alertar	2	2
Información de *logs*	2	2
Barrido de puertos	2	2

Tabla 2: Evaluación de Suricata y Snort

Desde el punto de vista funcional la evaluación de estas herramientas está nivelada. Por otra parte, atendiendo a los aspectos genéricos se determina que Snort es más antiguo que Suricata, posee una mayor base de usuarios, una documentación más amplia y es más fácil de usar. Además, con su rendimiento y utilización de recursos es un IDS adecuado para las redes de tráfico moderado de las instituciones que utilizarán Nova Servidores en sus sistemas de redes. Después de este análisis se decide seleccionar a Snort como IDS para la solución.

1.5 Herramientas para la centralización de *logs*

La administración y análisis de *logs* se ha tornado popular en cualquier sistema, debido a las facilidades y comodidades que brinda. En estos momentos se cuenta con numerosas herramientas que llevan a cabo este proceso, a continuación, se dan cita a algunas de las más usadas.

Logrep

Logrep es un *framework*[16] multiplataforma[17] seguro para la recopilación, extracción y presentación de información de varios archivos de registro. Cuenta con informes HTML, análisis multidimensional, páginas de vista general, comunicación SSH y gráficos, y soporta sistemas populares como *Squid, Postfix, Apache, Sendmail, syslog, ipchains, iptables, registros* de eventos *NT, Firewall-1, wtmp , Xferlog* y Pix.

Entre sus tantos beneficios se encuentran:

- Soporte para plataformas múltiples y formatos de *log*.

[16] Es un esquema (esqueleto, patrón) para el desarrollo e implementación de una aplicación.

[17] Está disponible para varios sistemas operativos.

- Mantiene copias comprimidas de *logs* en una ubicación central.
- Comunicación SSH segura entre el cliente y el servidor.
- Capaz de realizar un análisis multidimensional.
- Paquetes de instalación binario.
- De código abierto y altamente personalizable [21].

Logcheck

La herramienta *Logcheck* ayuda a detectar problemas automáticamente en los *logs* y envía los resultados periódicamente por *e-mail.* Soporta tres niveles de filtrado: "*Paranoid"* es para máquinas de alta seguridad ejecutando el menor número de servicios posible. No se debe usar si no se cuenta con la determinación suficiente para lidiar con sus mensajes verbosos. "Server" es el nivel predeterminado y contiene reglas para diferentes demonios. "*Workstation"* es para máquinas protegidas y filtra la mayoría de los mensajes. Las reglas para ignorar trabajan de manera aditiva. Las reglas de "*Paranoid"* están incluidas en el nivel "*Server*" y el nivel *"Workstation*" incluye las reglas de ambos: "*Paranoid*" y "*Server"*.

Los mensajes reportados se clasifican en tres capas, eventos del sistema, eventos de seguridad y alertas de ataque. La verbosidad de los eventos del sistema es controlada por el nivel que se elija, *paranoid*, *server* o *workstation*". Sin embargo, los eventos de seguridad y las alertas de ataque no se ven afectados por este problema [22].

Rsyslog

Rsyslog es una herramienta que no solo configura los *logs* de forma local, sino también acepta conexiones remotas con el fin de recibir *logs* de otros equipos y poder centralizarlos. La herramienta implementa el protocolo básico con una configuración flexible. Es un rápido sistema de procesamiento de registros de sistema. Ofrece un diseño modular de alto desempeño y niveles de seguridad apropiados. A diferencia de sus predecesores (sysklog y syslog), permite el ingreso de datos desde diversas fuentes, trasformación de datos y salida de resultados hacia varios destinos. Es lo suficientemente versátil y robusto para ser

utilizado en entornos empresariales y tan ligero y sencillo que permite utilizarlo también en sistemas pequeños.

Rsyslog permite almacenar las bitácoras en archivos de texto simple o bases de datos *MySQL* y PostgreSQL, utiliza TCP como protocolo de transporte y provee un control detallado de formato cola de procesamiento y capacidades de filtrado en cualquier parte de los mensajes. El *Rsyslog* es un *syslog* enfocado en la seguridad y confiabilidad, ofreciendo soporte para diversas operaciones tales como la demanda de almacenamiento en búfer [23].

Swatch

Esta herramienta permite leer los ficheros de registro de una vez o indicándole que vaya leyendo cada línea según se va añadiendo, además de permitir también la salida de distintos comandos.

Swatch está programada en *Perl*, y necesita algunos módulos disponibles de CPAN que durante el procedimiento de instalación te ayudará a descargar e instalar automáticamente.

La configuración de esta herramienta se realiza mediante patrones y grupos de acción, los patrones son expresiones regulares de *Perl* que además son compatibles con las expresiones regulares de *grep*. Cuando una línea coincida con un patrón, se ejecutará la acción o acciones asociadas a ese patrón.

Las acciones que se pueden realizar son las siguientes:

- **echo:** Muestra la línea por la salida estándar.
- **Bell:** Emite un pitido en el terminal en el que se está ejecutando *Swatch*.
- **Exec:** Ejecuta el comando.
- **Mail:** Manda un correo por línea o líneas que hayan coincidido con el patrón.
- **Write:** Escribe las líneas a uno o más usuarios.
- **Pipe:** Envía las líneas que hayan coincidido a un programa como su entrada estándar.
- **Throttle:** Permite indicar la frecuencia con la que debe ocurrir una determinada entrada que sea mostrada más de una vez [24].

Barnyard2

Barnyard2 es un intérprete de código abierto para los archivos binarios de salida de Snort. Su uso principal es permitir que Snort escriba al disco de una manera eficiente y dejar la tarea de analizar datos binarios en varios formatos a un proceso separado que no hará que Snort pierda el tráfico de red. Posee tres modos de operación:

- *Batch.*
- Continuo.
- Continuo con marcador.

En el modo *batch* barnyard2 procesará los archivos especificados explícitamente y dejará de ejecutarse, en continuo, barnyard2 comenzará con una ubicación para buscar un patrón de archivo especificado y continuará procesando nuevos datos (nuevos archivos de cola de impresión) según aparezcan, y en modo continuo con marcador, también usará un archivo de punto de control (o archivo waldo en el mundo de snort) para rastrear dónde está. En caso de que el proceso barnyard2 termine mientras un archivo waldo está en uso, se reanudará el procesamiento en la última entrada como se muestra en dicho archivo. El procesamiento de barnyard2 está controlado por dos tipos principales de directivas: procesadores de entrada y *plugins* de salida. Los procesadores de entrada leen información desde un formato específico (actualmente el módulo de salida *spo_unified2* de Snort) y los emiten en una de varias maneras [25]. Barnyard2 es capaz de escribir los *logs* generados por Snort en una base de datos MySQL, la cual puede estar ya creada o puede ser creada por la propia herramienta.

Elección de la herramienta de centralización

Al concluir el estudio de las herramientas de centralización se determinó usar barnyard2 ya que anteriormente se seleccionó Snort como IDS a utilizar en el trabajo y como se ya se mencionó, barnyard2 representa la herramienta nativa para el trabajo con los archivos de salida de Snort.

1.6 Metodología a utilizar

El proyecto a realizar es pequeño, de tiempo relativamente corto para su terminación y se utiliza más la comunicación cara a cara que la documentación por lo que lo correcto sería la utilización de una

metodología ágil. Estas metodologías intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en las personas y los resultados.

Entre las metodologías ágiles más usadas se encuentran *Extrem Programing (XP), Scrum, Crystal Clear, Agile Unified Process (AUP).* En la Universidad de Ciencias Informáticas (UCI) se convergió el uso de una sola metodología para cubrir las particularidades de cada uno de los proyectos. Este proceso se llevó a cabo escogiendo una metodología ya existente para ser adaptada a lo que ya la Universidad ha estado proponiendo como ciclo de vida de los proyectos, sin alejarse de lo que hasta el momento se ha trabajado e introducir la menor cantidad de cambios posibles. Lo cual fue una variación que se le realizó a AUP, y es la metodología utilizada en este trabajo.

1.7 Herramientas y tecnologías

La selección acertada de herramientas y tecnologías para emplear en el proceso de desarrollo, apoyado en el dominio que se tenga de las mismas, es crucial para la obtención en tiempo de un producto de calidad. Con esa idea presente, se hizo uso de los recursos a continuación:

1.7.1 Modelado

El lenguaje de modelado **UML**[18] en su versión 2.5, permite comunicar la estructura de un sistema complejo, especificar el comportamiento deseado del sistema, comprender mejor lo que se está construyendo y a la vez descubrir oportunidades de simplificación y reutilización. Se basa en el hecho de que un modelo es la simplificación de la realidad [26].

Visual Paradigm se selecciona como herramienta CASE[19] de modelado, que utiliza UML para la completa representación de las etapas por las que transita un producto de software. Este permite la realización de una amplia gama de diagramas como: casos de uso, de actividades, de despliegue, entre otros, así como la generación de código fuente desde los mismos y la documentación asociada al proceso que esté siendo modelado.

[18] Lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

[19] Ingeniería de Software Asistida por Computadora (del inglés *Computer Aided Software Engineering*). Aplicaciones destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo del mismo en términos de tiempo y dinero.

1.7.2 Lenguaje de programación.

Seleccionar el lenguaje de programación es uno de los aspectos de más peso en los proyectos, debido a que se deben tener una gran cantidad de factores antes de elegir, por ejemplo: grado de familiaridad con el lenguaje del equipo de desarrollo, IDE que lo soporta, eficiencia y consumo de memoria, soporte del lenguaje para poder programar la aplicación deseada, y se podría seguir con una innumerable lista de factores a tener en cuenta. Existen dos familias de lenguajes de programación, los lenguajes de bajo nivel y los lenguajes de alto nivel. Los lenguajes de bajo nivel poseen una sintaxis muy cercana al Lenguaje de Máquina, el cual es muy difícil de interpretar por los seres humanos, mientras que los lenguajes de alto nivel ofrecen una sintaxis muy cercana al lenguaje de los seres humanos. Entre los lenguajes de programación de alto nivel más populares se encuentran el C, C++, Java, C#, Python, D, entre otros.

Python

Debido a su integración por defecto a los sistemas de código abierto y la facilidad para su comprensión se escoge *Python* en su versión 3.4.3 como lenguaje de programación. *Python* es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. Su elegante sintaxis y tipado dinámico junto con su naturaleza interpretada, hacen de este un lenguaje ideal para *scripting* y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

Algunos casos de éxito en el uso de *Python* son Google, Yahoo, la NASA, Industrias *Light* & *Magic* y todas las distribuciones Linux, en las que cada vez representa un tanto por ciento mayor de los programas disponibles. Es administrado por la *Python* Software Foundation, una organización sin fines de lucro orientada al avance de tecnologías de código abierto relacionadas con dicho lenguaje de programación. Posee una licencia de código abierto, denominada *Python Software Foundation License*[20] [27].

1.7.3 Entorno de desarrollo

Como entorno de desarrollo Integrado se utiliza ***Pycharm*** en la propuesta de solución. ***Pycharm*** es un entorno de desarrollo integrado multiplataforma utilizado para desarrollar en el lenguaje de programación

[20] Licencia usada para la distribución de software del proyecto *Python*

Python. Provee funcionalidades que permiten una experiencia única y aumentan en gran medida la productividad:

- Completamiento de código de manera inteligente y señalamiento de errores con reparación de los mismos de forma automatizada.
- Integración con marcos de trabajo de desarrollo web modernos como Django.
- Depurador integrado y herramientas de pruebas.
- Soporte para bases de datos e integración con sistemas de control de versiones.

En adición a *Python*, PyCharm soporta *JavaScript*, *CoffeeScript, TypeScript, HTML/CSS*, *Cython,* lenguajes de plantilla, *AngularJS* y *Node.js* [28].

1.7.4 Sistema Gestor de Base de Datos

MySQL es un sistema de gestión de base de datos relacional (RDBMS), por sus siglas en inglés, de código abierto, basado en lenguaje de consulta estructurado (SQL), por sus siglas en ingles. Se ejecuta prácticamente en todas las plataformas, incluyendo Linux, Unix, Windows. A pesar de que se puede utilizar en una amplia gama de aplicaciones, *MySQL* se asocia más con las aplicaciones basadas en la web y la publicación en línea. Las base de datos *MySQL* son relacionales lo que quiere decir que almacenan los datos en tablas separadas, También son rápidas, de confianza y fácil de usar.

En el trabajo se decidió el uso de *MySQL* ya que además de sus ventajas se torna obligatorio usarlo debido a la utilización de barnyard2 la cual, como se dijo anteriormente, escribe los *logs* de Snort en una base de datos *MySQL*.

1.7.5 Herramienta de virtualización

Para la instalación de las máquinas virtuales que simularán las computadoras donde los IDS serán instalados en el sistema de red se utiliza la herramienta *Oracle VM Virtual Box* el cual es un software de virtualización[21] multiplataforma para arquitecturas x86 y X64 [29]. Virtual box se desarrolla activamente

[21] Se puede crear y ejecutar varias máquinas virtuales con corriendo diferentes sistemas operativos, en una misma computadora al mismo tiempo.

con los lanzamientos frecuentes apoyado por los sistemas operativos que virtualiza y las plataformas que soporta. La empresa Oracle es la encargada de que se cumpla con sus criterios de calidad profesional.

Conclusiones Parciales

El estudio de las alternativas a la propuesta de solución evidenció la necesidad e importancia de la centralización y visualización de los registros de eventos generados por el IDS para Nova Servidores.

La selección de ELK mediante el método QSOS para el proceso de búsqueda, recolección y visualización de los *logs*, MySQL como gestor de base de datos y Python como lenguaje de programación demostró la capacidad de esas modernas y poderosas tecnologías, en aras de lograr el desarrollo de una herramienta de alta calidad, guiada por la metodología AUP en su variante para la UCI.

Capítulo 2. Análisis y Diseño

Introducción

En el presente capítulo se establecen las principales características y funcionalidades con que contará la solución ya sean requisitos funcionales o no funcionales. Se generan los diagramas correspondientes a la fase de ejecución que propone la metodología AUP en su variante para la UCI. Además, se describen los patrones de diseño y arquitectura que se utilizarán.

2.1 Propuesta de Solución

Para darle solución a la situación problemática se propone la instalación de Snort junto con barnyard2 en las computadoras del sistema de red, las cuales estarán conectadas a un servidor que contendrá la base de datos donde barnyard2 enviará los *logs* generados por Snort realizando así el proceso de centralización. El servidor también contendrá el *stack ELK* por lo que *Logstash* extraerá los *logs* de la base de datos y los enviará a *Elasticsearch* para que este realice la indexación de los *logs* y posteriormente Kibanalos extraerá de *Elasticsearch* para realizar el proceso de visualización.

2.1.1 Modelo de dominio

A continuación se describe el funcionamiento del sistema a partir del modelo de dominio. En dicho modelo se detalla el proceso de centralización y visualización de los *logs* del IDS en Nova Servidores (*Ver ilustración 1*).

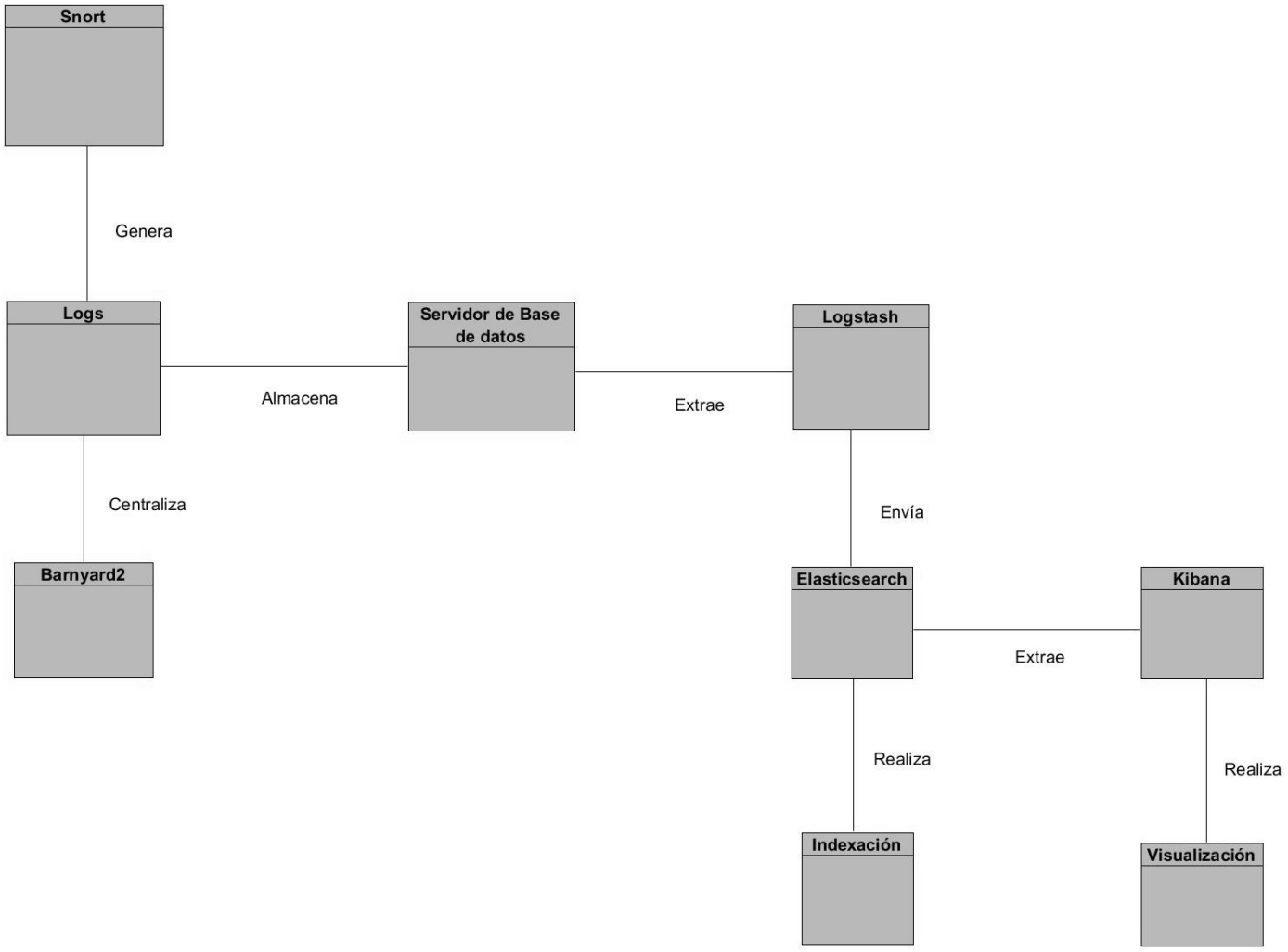


Ilustración 1: Modelo de dominio

2.1.2 Glosario de conceptos del Modelo de dominio

Snort: Es el IDS instalado en las diferentes computadoras.

Barnyard2: Herramienta que escribirá los logs generados por snort en una base de datos y los centralizará.

Logs: Registros de eventos generados por Snort (representan las alertas de Snort).

Servidor de Base de datos: Computadora donde se encontrará la base de datos que almacenará los logs.

Logstash: Tubería de datos que se encargará de extraer los logs de la base de datos y enviarlos a *elasticsearch*.

Elasticsearch: Herramienta que indexará los logs.

Kibana: Se encargará de visualizar los *logs* indexados por *elasticsearch*.

2.2.3 Diagrama de Paquetes del módulo IDS.

Para la descripción de los paquetes incluidos en el módulo IDS a desarrollar se elaboró el **diagrama de paquetes** del mismo (*Ver ilustración 2*).

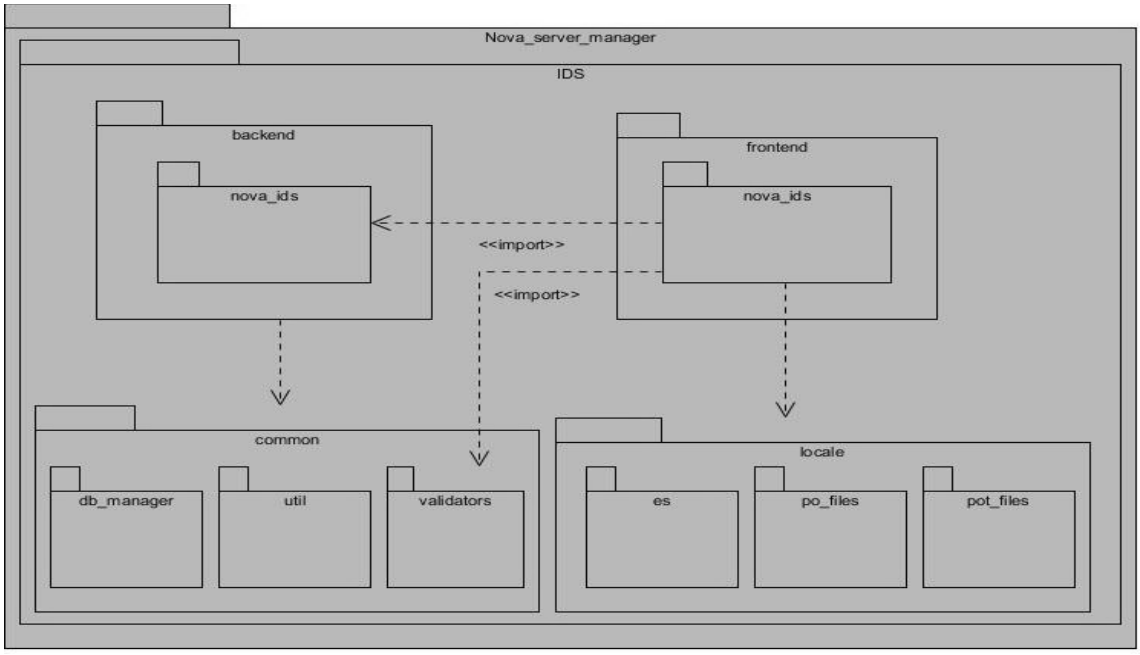


Ilustración 2: Diagrama de paquetes

- ***Nova_server_manager:*** es el gestor de administración de Nova Servidores. Se encarga de la instalación, configuración y desinstalación de los servicios para servidores con que cuenta Nova servidores 6.0. Posee varios paquetes, los cuales abarcan las funcionalidades que posibilitan el uso de los servicios contemplados dentro de la arquitectura del sistema. Se agrupan las funcionalidades utilizables y configurables por la herramienta Nova-server-manager. Se maneja también todo lo relacionado con las validaciones de configuraciones, las utilidades e internacionalización de la herramienta y las vistas con las que interactuarán los usuarios.
- **IDS*:*** módulo para la creación del IDS. Permitirá la configuración y administración del IDS.
- ***Frontend:*** se encontrarán definidas todas las clases de las vistas del IDS.
- ***Backend:*** es el paquete donde se encontrarán definidas las clases controladoras del IDS.
- ***Locale:*** paquete de internacionalización del sistema.
- ***Common:*** se encuentran definidos los modelos de datos y los validadores

2.3 Levantamiento de requisitos

El levantamiento de requisitos cumple un papel esencial en el proceso de desarrollo de un producto de software, se orienta a la definición de lo que se desea producir, describiendo con claridad, sin imprecisiones, en forma estable y compacta. Define el comportamiento que deberá tener el componente, minimizando la posibilidad de errores que puedan ocurrir relacionados al desarrollo del mismo, pues se especifica de forma clara lo que el cliente desea. Los requisitos se dividen en dos grupos: requisitos funcionales y requisitos no funcionales.

2.3.1 Requisitos funcionales

Un requisito funcional define una función del software o sus componentes; pueden ser cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir [30].

Código	Nombre	Prioridad
RF1	Iniciar Snort	Alta
RF2	Detener Snort	Alta
RF3	Iniciar Barnyard2	Alta
RF4	Detener Barnyard2	Alta
RF5	Configurar base de datos	Alta
RF6	Visualizar logs	Alta
RF7	Visualizar logs dependiendo del tipo de ataque	Baja
RF8	Visualizar logs dependiendo de la fecha.	Baja
RF9	Visualizar logs dependiendo de la dirección IP.	Baja

Tabla 1: Requisitos funcionales

2.3.2 Requisitos no funcionales

Un requisito no funcional especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales. Sirven de apoyo a los requisitos funcionales [30].

Interfaz

El sistema deberá presentar facilidades al usuario para el manejo de la información, para lo cual se utiliza la interfaz gráfica de la herramienta *Nova_Server_Manager* para la gestión del módulo de administración del IDS la cual es sencilla y fácil de usar y así, propiciar un buen entendimiento al usuario. También se utiliza la interfaz proporcionada por Kibana para la visualización de *logs*, la cual brinda muchas comodidades como permitir mostrar *logs* de acuerdo al filtrado de sus campos.

Restricciones de implementación:

- Utilizar como lenguaje de programación *Python.*
- El estilo de programación debe ser *lowerCaseCamel*.

Restricciones de Software:

- Se necesita la instalación del sistema operativo Nova 6.0 en su versión para servidores.
- Se necesita la instalación de un entorno de ejecución de java para la utilización del *stack* ELK.

Restricciones de Hardware:

Para la utilización eficiente de las máquinas virtuales que se deben instalar se requiere de una computadora con al menos 4gb de RAM.

2.4 Historias de usuarios

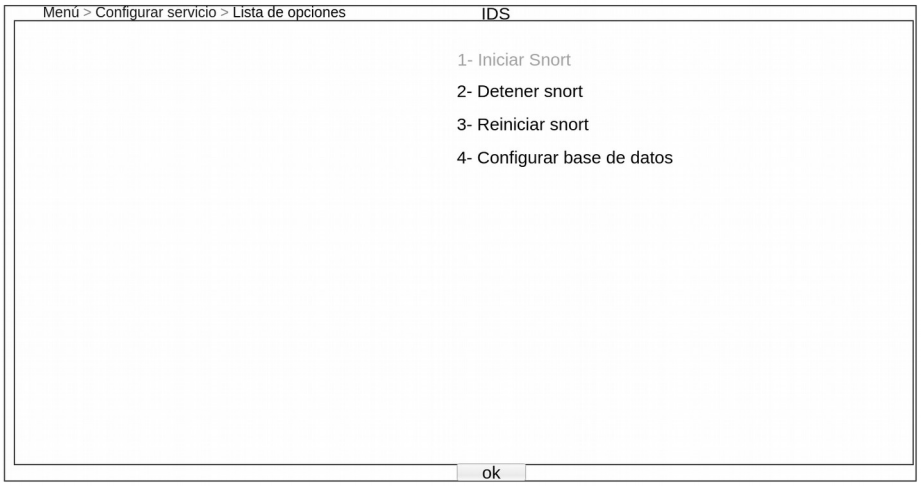
A partir de que la metodología utilizada se proponen tres variantes a utilizar en los proyectos (CUN, DPN o MC), existen tres formas de encapsular los requisitos (CUS, HU, DRP) y surgen cuatro escenarios para modelar el sistema en los proyectos, manteniendo en dos de ellos el MC.

En el caso de la solución propuesta se tiene un modelo bien definido y el cliente estará acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Teniendo en cuenta estas características se decidió la adopción del escenario que utiliza las historias de usuario para encapsular los requisitos funcionales.

Las historias de usuario administran los requisitos sin la elaboración de una gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las mismas son escritas utilizando el lenguaje común del usuario y son utilizadas en las metodologías ágiles para la especificación de requisito [31].

Para describir los requisitos funcionales de la propuesta de solución se definió una historia de usuario para cada uno de ellos.

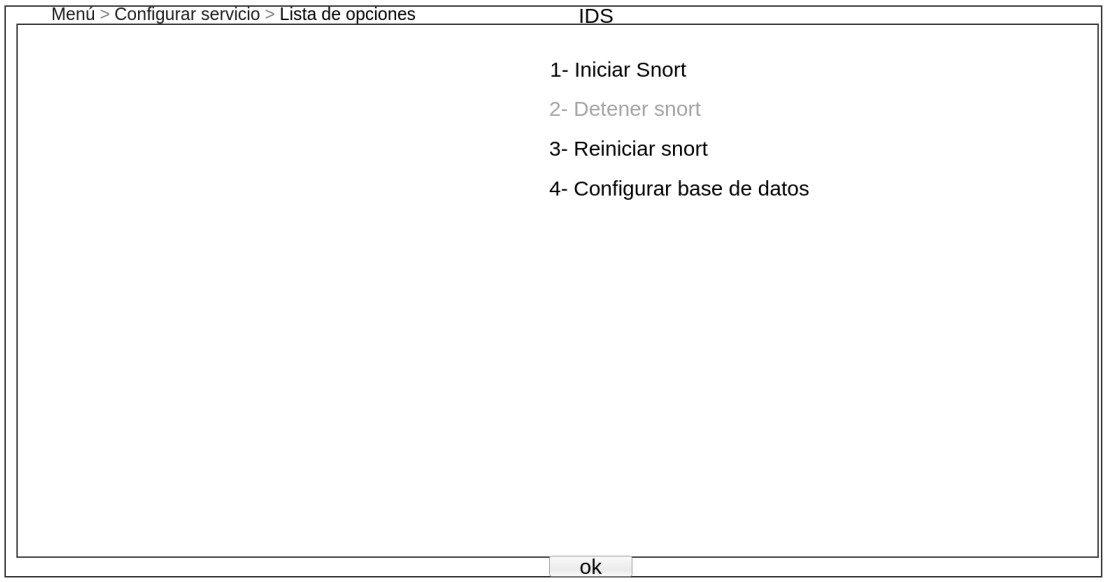
Centralización y Visualización de *logs* del IDS para Nova Servidores

Número 1:	**Nombre del Requisito:** Iniciar Snort
Programador: Rubén Lage Laricheva	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 días
Riesgo en Desarrollo: Interrupción del fluido eléctrico.	**Tiempo Real**: 1 Semana
Descripción: La aplicación debe permitirle al usuario ejecutar el snort para que empiece su servicio como IDS. Esta funcionalidad se encontrará en la interfaz principal de la aplicación	
Prototipo de Interfaz:	

Prototipo de interfaz:	

Tabla 2: Historia de usuario. Iniciar Snort

Centralización y Visualización de *logs* del IDS para Nova Servidores

Numero 2:	**Nombre del Requisito**: Detener Snort
Prioridad: Alta	Iteración.
Programador: Rubén Lage Laricheva	**Tiempo estimado:** 3 días
Descripción: La aplicación debe permitir detener snort, esta funcionalidad se activa después de que el usuario inicia snort y aparecerá en la interfaz principal de la aplicación.	
Prototipo de interfaz:	

Prototipo de interfaz:	

Tabla 3: Historia de usuario. Detener Snort

Número 3:	**Nombre del Requisito:** Iniciar Barnyard2
Programador: Rubén Lage Laricheva	**Iteración: 1**
Prioridad: Alta	**Tiempo estimado:** 5 días
Descripción: Esta funcionalidad se realizará automáticamente cuando el usuario decida iniciar snort, por lo que no aparecerá en la interfaz y permitirá al usuario iniciar la herramienta barnyard2 para escribir los logs de snort en la base de datos MySQL.	**Tiempo Real:** 1 semana

Tabla 4: Historia de usuario. Iniciar Barnyard2

Número 4:	**Nombre del Requisito:** Detener Barnayard2
Programador: Ruben Lage Laricheva.	**Iteración: 1**
Prioridad: Alta	**Tiempo Estimado:** 3 días
Descripción: Esta funcionalidad se realizará automáticamente cuando el usuario decida detener snort, por lo que no aparecerá en la interfaz y permitirá al usuario detener la herramienta barnyard2.	**Tiempo Real:** 3 días

Tabla 5: Historia de usuario. Detener Barnyard2

Centralización y Visualización de *logs* del IDS para Nova Servidores

Numero 6:	**Nombre del Requisito:** Visualizar logs.
Programador: Rubén Lage Laricheva	**Iteración: 1**
Prioridad: Alta	**Tiempo Estimado:** 7 días.
Descripción: La aplicación le permitirá al usuario ver los logs generados por Snort almacenados por Barnyard2 en la base de datos mysql. Esta funcionalidad se activará después de que el usuario haya iniciado snort y la interfaz la brindará Kibana.	**Tiempo real:** 10 días.

Prototipo de interfaz:	

Tabla 6: Historia de usuario. Visualizar logs almacenado en la base de datos

2.5 Diseño

El papel del diseño en el ciclo de vida de un software es facilitar la comprensión de su funcionamiento y proveer una representación o modelo del mismo con el propósito de definirlo con los suficientes detalles como para permitir su realización física. El modelo de diseño provee una representación arquitectónica de software que sirva de punto de partida para las tareas de implementación, dando al traste con los requisitos del sistema.

2.5.1 Arquitectura Modelo-Vista-Controlador

El estándar IEEE[22] define la arquitectura de software como la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución [32]. Según Roger Pressman: “En su forma mas simple, la arquitectura del software es la estructura u organización de los componentes del programa, la manera en que estos interactúan y la

[22] *Institute of Electrical and Electronic Engineers.* Traducido al español Instituto de Ingeniería Eléctrica y Electrónica.

estructura de datos que utilizan [33].”

O sea, la arquitectura de software no es más que una forma de representar sistemas mediante el uso de la abstracción, de forma que aporte el más alto nivel de comprensión de los mismos. Esta representación incluye los componentes fundamentales del software, su comportamiento y formas de interacción para satisfacer los requisitos del sistema.

Para el desarrollo del módulo para gestionar el IDS de Nova Servidores y su integración a la herramienta Nova-Server-Manager se decidió escoger la arquitectura Modelo-Vista-Controlador, ya que es la Arquitectura que utiliza Nova Server Manager para sus módulos.

Esta arquitectura separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente modelo se encarga de manejar los datos asociados al sistema y las operaciones asociadas a esos datos (Paquete ***Common***). El componente Vista define y gestiona como se presentan esos datos al usuario (Paquete ***Frontend***). El componente controlador dirige la interacción del usuario y pasa estas interacciones a Vista y Modelo (Paquete ***Backend***).

2.5.2 Patrones de diseño

Un patrón de diseño de software se presenta como una solución reutilizable a un problema recurrente en el desarrollo. Su reutilización reduce el esfuerzo, tiempo y costos empleados durante el diseño de software. Los patrones de diseño capturan la estructura y dinámica de una solución que se repite múltiples veces durante el desarrollo de diferentes aplicaciones, generalmente, en un contexto o dominio determinado [34].

Para la propuesta de solución se hizo uso de varios patrones *Patrones Generales de Software para asignar Responsabilidad (GRASP).*

El patrón **Alta cohesión** plantea la asignación de responsabilidades de manera que la cohesión se mantenga alta, siendo la cohesión el parámetro que mide el grado en que están relacionadas las responsabilidades de una clase [35]. En la solución este patrón se pone de manifiesto en la estructura de módulos los cuales tienen tareas y objetivos funcionales únicos dentro de la aplicación, así como en la creación de clases donde la información que almacenan son coherentes y altamente relacionadas con la

clase.

Bajo acoplamiento expone la necesidad de asignar responsabilidades de manera que el acoplamiento sea bajo, lo cual permite crear clases más independientes, más reutilizables, lo que implica mayor productividad [35]. Este patrón se utiliza en la solución en la separación funcional de las clases y partes de la aplicación permitiendo al código ser más fácil de mantener y que en el momento en que se requieran cambios las demás partes se afecten lo menos posible.

Además de los mencionados anteriormente se utilizó el patrón ***singleton*** perteneciente a la llamada *Banda de los Cuatro (GOF* por sus siglas en inglés).

Singleton**:** Este patrón es de tipo “creación a nivel de objetos” y su propósito es garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma [36]. La herramienta Nova_Manager utiliza este patrón para asegurarse que solo se esté trabajando sobre una misma instancia de la interfaz del Dialog Manager la cual crea la interfaz en la consola y se encuentra en frontend/ui/dialogManager.py, class DialogManager(Singleton).

2.6 Diagrama de clases

Se muestra en la siguiente ilustración el diagrama de clases del diseño del módulo IDS, que cuenta con dos clases principales, *Main* en el paquete *frontend* que proporciona la vista del sistema y su relación de importación con la clase *Main* en el paquete *backend* que es la clase controladora del sistema *(Ver ilustración 3*).

Ilustración 3: Diagrama de clases de la solución

Conclusiones parciales

La elaboración del modelo de domino junto con la propuesta de solución permitió esclarecer conceptos claves en cuanto a la estructura y funcionamiento de la aplicación, así como la obtención de los requisitos no funcionales y funcionales, descritos en las historias de usuario. La arquitectura modelo-vista-controlador definida permitió la utilización de una interfaz que posibilite realizar las operaciones pertinentes para la solución propuesta. Además, la utilización de patrones de diseño GRASP y GOF garantizaron la organización del código y el empleo de buenas prácticas en el proceso de codificación.

Capítulo 3. Implementación y Prueba

Introducción

Para materializar la propuesta de solución y cumplir con los requisitos obtenidos al comenzar la investigación, se lleva a cabo la fase de implementación y prueba. En esta fase se recogen los estándares de codificación que deben ser empleados en el desarrollo del entorno colaborativo, el diagrama de despliegue donde se observan las dependencias lógicas que existen entre los componentes del software y los nodos necesarios para la puesta en marcha del sistema. Finalmente se muestran las pruebas realizadas para validar el correcto funcionamiento de la propuesta de solución.

3.1 Estándar de codificación

La existencia de un estándar de codificación es necesario pues debe existir un estilo consistente a lo largo de toda la aplicación y no el preferido por cada programador involucrado. Esto posibilita que se pueden añadir nuevas funcionalidades, modificar las ya existentes o depurar errores con gran facilidad y obtener un código con mayor legibilidad. A continuación, se define el estándar a utilizar en la implementación del entorno colaborativo:

- Los nombres de las variables, clases y métodos serán en inglés.
- Para nombrar identificadores se empleará el estilo *lowerCaseCamel*[23]
- Se utilizarán 4 espacios para la identación[24] del código.
- No mezclar tabulaciones con espacios, el método de identación más popular en *Python* es el uso de espacios y el segundo más utilizado es con tabulaciones, pero no deben ser mezclados uno con el otro.
- El tamaño de línea máximo será de 79 caracteres para evitar la necesidad de realizar un

[23] Estilo de escritura que se aplica a frases o palabras compuestas. La primera letra de cada una de las palabras es mayúscula con la excepción de que la primera letra es minúscula

[24] Significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izq uierdo y mejor distinguirlo del texto adyacente

desplazamiento horizontal para visualizar el código fuente.

- Se hará uso de comentarios en aquellas funcionalidades de mayor complejidad.
- Se debe declarar cada variable en una línea diferente y de esta forma se puede comentar cada variable por separado.
- Las definiciones de métodos dentro de una clase son separados por una línea en blanco.

3.2 Diagrama de despliegue

Un diagrama de despliegue consiste en un diagrama estructurado que muestra la arquitectura del sistema desde el punto de vista del despliegue o distribución de los artefactos del *software* en los destinos de despliegue; definiendo a los artefactos como representaciones de elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo. Ejemplos de artefactos son archivos ejecutables, bibliotecas, archivos, esquemas de BD y archivos de configuración. Cuando se refiere a destino de despliegue se hace referencia a un nodo que es o bien un dispositivo de hardware o bien un entorno de ejecución de *software.* La siguiente imagen representa el diagrama de despliegue de la solución (*Ver ilustración 4*)

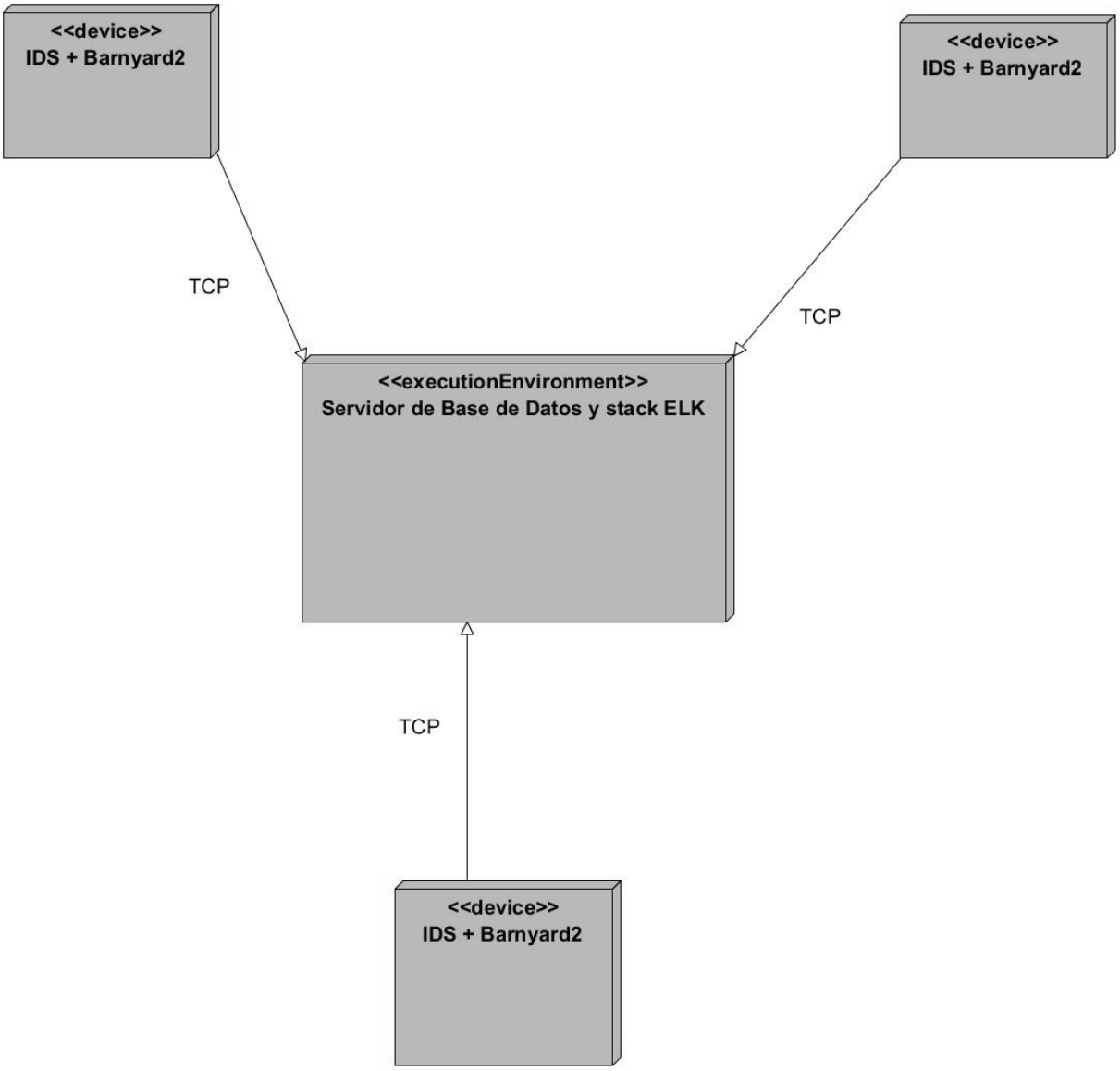


Ilustración 4: Diagrama de despliegue

IDS: Se refiere a la estación de trabajo que tendrá instalado un IDS la cual podrá conectarse al servidor de

base de datos mediante el archivo de configuración de la herramienta *barnyard2.*

Barnyard2: Herramienta que almacena los *logs* del IDS en la base de datos.

Servidor de Base de datos y *stack* ELK: Es la estación de trabajo que contiene la base de datos que centraliza los logs de los diferentes IDS y el *stack* ELK para su posterior visualización.

3.3 Interfaz para la visualización de los logs

Esta interfaz la provee *Kibana,* y permitirá elaborar potentes gráficas en función de los diferentes campos que los *logs* incluyen (dirección ip, fecha, tipo de alerta, tipo de ataque).

3.3.1 Gráfica de área

Kibana permite representar los datos indexados por *elasticsearch* mediante diferentes tipos de gráficas, a continuación se muestra una imagen de una gráfica de área representando la cantidad de *logs* en función de la fecha (*Ver ilustración 5*):

Ilustración 5: Gráfico de área

3.3.2 Gráfica de barras verticales

A continuación, se muestra un ejemplo de una gráfica de barras verticales la cual está representando la cantidad de logs en función del tipo de alerta (*Ver ilustración 6*).

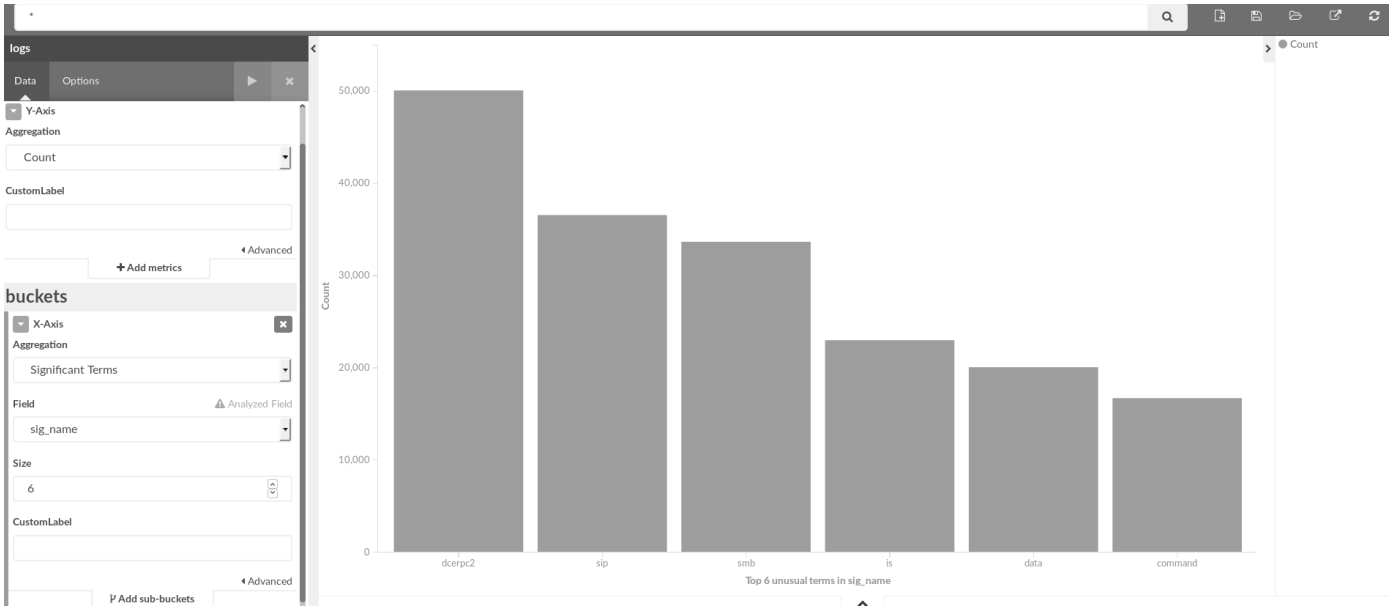


Ilustración 6: Gráfica de barras verticales (cantidad de logs por tipo de alertas)

3.3.3 Gráfica de línea

Otras de los tipo de gráficas que permite mostrar *Kibana* es la gráfica de línea, a continuación se presenta este tipo de gráfica ilustrando la cantidad de logs en función del ip destino (*Ver ilustración 7*):

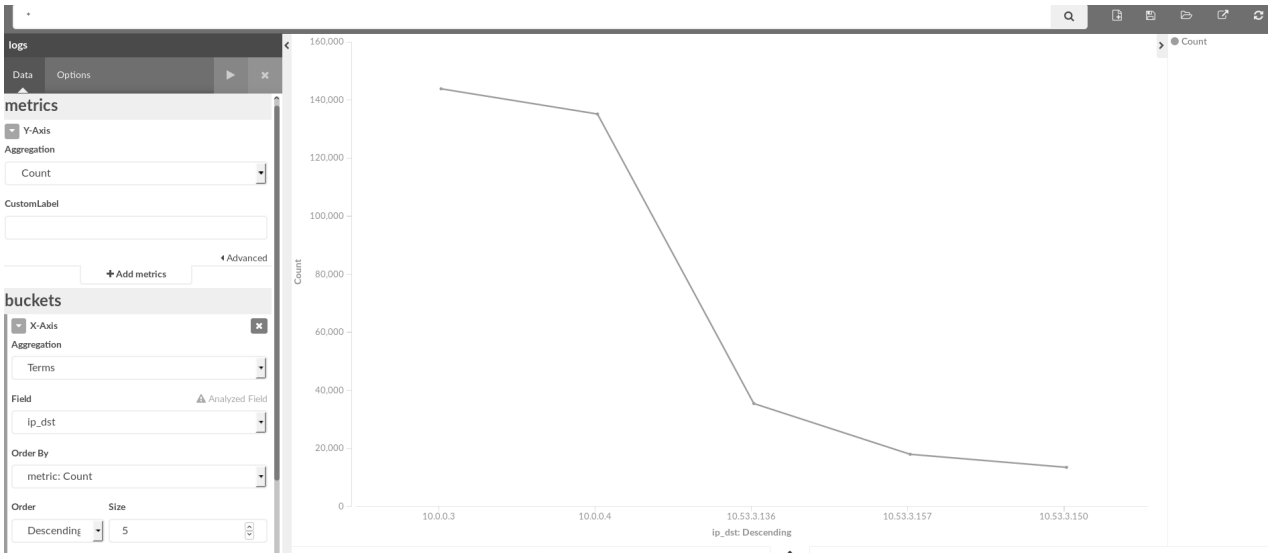


Ilustración 7: Gráfica de línea(cantidad de logs por ip de destino)

3.3.4 Gráfica de pastel

La última gráfica que se ejemplificará es la gráfica de pastel, este ejemplo ilustra la cantidad de *logs* en función del tipo de ataque (*Ver ilustración 8*).

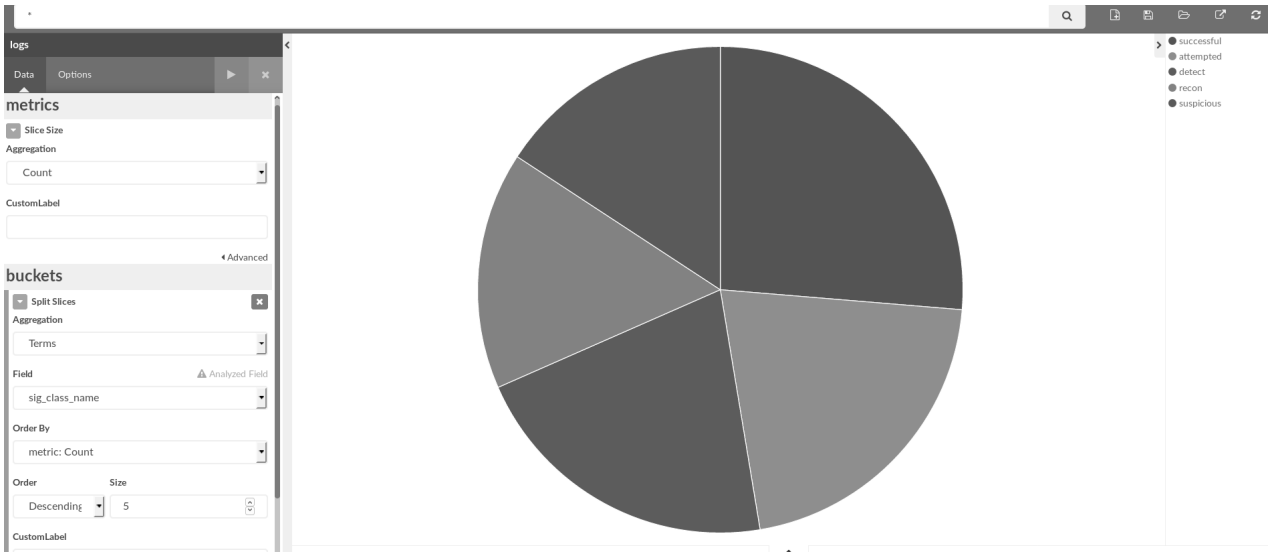


Ilustración 8: Gráfica de pastel(cantidad de logs por tipo de ataque)

3.4 Pruebas de software

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan la excelencia y el desempeño de un software, involucra las operaciones del sistema bajo condiciones controladas y evaluando los resultados. Las técnicas para encontrar problemas en un programa son variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad [37].

3.4.1 Pruebas a realizar

Funcionales

Se escoge este nivel de prueba ya que lo que se desea es la verificación de las funcionalidades del sistema a través de las interfaces y su relación con los demás componentes del sistema.

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 [37].

Para no caer en este absurdo, el ingeniero del software debe anticiparse a posibles problemas de la interfaz:

- Diseñar ruta de manejo de errores que prueben toda la información proveniente de otros elementos del sistema.
- Aplicar una serie de pruebas que simulen datos incorrectos u otros posibles errores en la interfaz de software.
- Registrar los resultados de la prueba como evidencia en el caso de que se culpe.

Pruebas de aceptación

Se escoge la prueba de aceptación ya que es necesario, para la validación de la solución, que el cliente esté de acuerdo con el funcionamiento del sistema desarrollado y así pueda emitir la carta de aceptación que demuestra la conformidad del cliente con la solución. Como técnica se escoge las pruebas alfa esta se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario. Las pruebas alfa se llevan a cabo en un entorno controlado. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados [37].

Unitarias

La prueba de unidad se concentra 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 a nivel de componentes, se prueban importantes caminos de control para descubrir errores dentro de los límites del módulo. 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 [33].

Integración

La prueba de integración es una técnica simétrica para construir la arquitectura del software mientras, al mismo tiempo, se aplican las pruebas para descubrir errores asociados con la interfaz. El objetivo de tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño. La estrategia para realizar la prueba de integración seleccionada por el autor de la investigación científica es la integración ascendente [33].

La estrategia de integración ascendente como su nombre lo indica, empieza la construcción y la prueba con componentes de los niveles más bajos de la estructura del programa [33].

3.4.2 Método de prueba

Caja negra

Se escoge el método de caja negra ya que los niveles de pruebas escogidos pretenden probar el funcionamiento de la interfaz y de las funcionalidades del sistema, para esto es necesario este método ya que permite comprobar el comportamiento del software a través de los requisitos funcionales [37]. Las pruebas de caja negra pretenden encontrar estos tipos de errores:

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

Caja blanca

La prueba de caja blanca, en ocasiones llamada prueba de caja de cristal, 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. Al emplear los métodos de prueba de caja blanca, el ingeniero del software podrá derivar casos de prueba que [33] :

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

3.4.3 Técnica de prueba Partición equivalente

Características

- Divide el dominio de entrada de un programa en clases de datos, a partir de las cuales pueden

derivarse casos de prueba.

- Descubre clases de errores, que, de otra manera, requeriría la ejecución de muchos casos antes de que se observe el error general.
- Reducir al máximo el total de casos de prueba que deben desarrollarse.

El diseño consiste en:

- Identificar clases de equivalencia.
- Crear los casos de prueba.

Clase de equivalencia

Conjunto de estados válidos o inválidos para condiciones de entrada. Las clases de equivalencia se

definen de acuerdo a 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 específico, 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 otra no válida.
4. Si una condición de entrada es booleana, se define una clase de equivalencia válida y otra no válida [38].

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 ejecutan cada instrucción del programa por lo menos una vez durante la prueba [33].

3.5 Aplicación de pruebas

3.5.1 Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de prueba ejecutables para automatizar las pruebas [33].

Las pruebas unitarias fueron realizadas mediante el método de caja blanca y la técnica camino básico. Las unidades de pruebas más pequeñas son las operaciones dentro de la clase. A continuación, se realiza la técnica del camino básico, al método *SnortStatus()* en la clase *Main* en el paquete *backend (Ver*

ilustración 9).

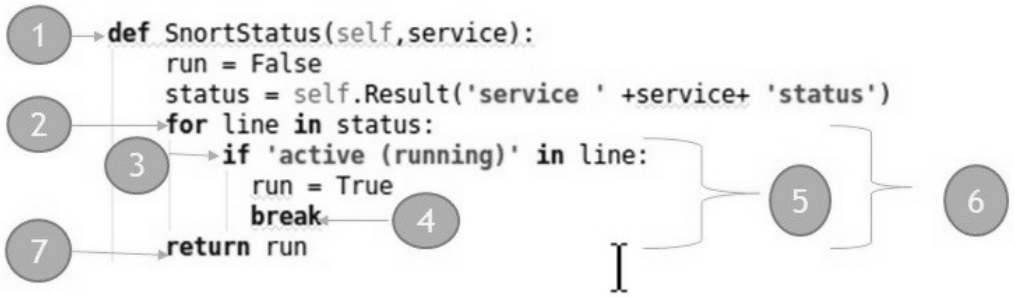


Ilustración 9: Prueba de camino básico

Luego de numerar las líneas de código, se diseña la gráfica del programa que describe el flujo de control lógico empleando nodos y aristas (Ver ilustración 10).

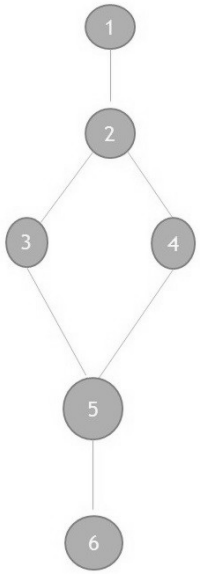


Ilustración 10: Grafo de flujo.
Elaboración propia

A partir del grafo obtenido con 7 nodos y 7 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_aristas – cantidad_nodos + 2

V(G) = 7– 7 + 2 = 2

Un camino independiente es cualquier camino del programa que introduce al menos un nuevo conjunto de sentencias de proceso o una nueva condición [33]. La cantidad de caminos independientes se establece por la complejidad ciclomática, por tanto, se identifican 2 caminos como se muestra en la siguiente tabla.

No	Camino
1	1-2-3-5-6-7
2	1-2-4-5-6-7

Tabla 7: Listado de caminos independientes.

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 [33]. A continuación, se muestran los casos de pruebas para cada camino independiente.

Caso de prueba de unidad	
No. Camino: 1	**Camino:** 1-2-3-5-6-7
Nombre de la persona que realiza la prueba: Ruben Lage Laricheva.	
Descripción de la prueba: Indica el estado del servicio activo.	
Entrada: Si.	
Resultado esperado: Devuelve un valor booleano: verdadero.	
Evaluación de la prueba: Satisfactoria.	

Tabla 8: Caso de prueba de unidad del camino número 1

Caso de prueba de unidad	
No. Camino: 2	**Camino:** 1-2-3-5-6-7
Nombre de la persona que realiza la prueba: Ruben Lage Laricheva.	
Descripción de la prueba: Indica el estado del servicio no activo.	
Entrada: Si.	
Resultado esperado: Devuelve un valor booleano: falso.	
Evaluación de la prueba: Satisfactoria.	

Tabla 9: Caso de prueba de unidad del camino número 2

Resultado de las pruebas unitarias

Fueron realizadas dos iteraciones mediante el método de caja blanca mediante la técnica de camino básico donde fue encontrado un error de llamada a métodos.

Pruebas Funcionales

Las pruebas funcionales fueron realizadas mediante en el método de caja negra mediante la técnica partición equivalente. A continuación, se muestran el diseño de Casos de Pruebas basados en Historias de Usuarios.

Diseño de Casos de Prueba basados en historias de usuarios:

Caso de prueba 1. Iniciar snort

Escenario	Descripción	V1 Selección	Respuesta del sistema	Flujo central
Ec 1.1 Iniciar snort	Se inicia el servicio snort en el sistema.	Selecciona la primera opción “Start snort”. Selecciona la opción “Aceptar”	El sistema muestra un mensaje de confirmación: “The service has been started.”	1-El usuario selecciona la opción “Configure service” y presiona “aceptar”. 2-El usuario selecciona la opción “Start snort” y presiona “Aceptar”.
Ec 1.2 No iniciar snort	No se inicia el servicio snort en el sistema	Selecciona la primera opción “Start snort”. Selecciona la opción “Aceptar”	El sistema muestra el mensaje de error predefinido que se muestra en la consola.	

Tabla 10: Descripción de los escenarios del caso de prueba “Iniciar snort”.

Caso de prueba 2. Detener snort.

Escenario	Descripción	V2 Selección	Respuesta del sistema	Flujo central
Ec 2.1 Detener snort	Se detiene el servicio snort en el sistema.	Selecciona la segunda opción “Stop snort”. Selecciona la opción “Aceptar”.	El sistema muestra un mensaje de confirmación: “The service has been stoped”.	1-El usuario selecciona la opción “Configure service” y presiona “aceptar”. 2-El usuario selecciona la opción “Stop snort” y presiona “Aceptar”.
Ec 2.2 No detener snort.	No se detiene el servicio snort.	Selecciona la primera opción “Stop snort”. Selecciona la opción “Aceptar”.	El sistema muestra el mensaje de error predefinido que se muestra en la consola.	

Tabla 11: Descripción de los escenarios del caso de prueba “Detener snort”.

Caso de prueba 4. Visualizar logs

Escenario	Descripción	V5 Selección	Respuesta del sistema	Flujo central
Ec 5.1 Visualizar logs	Se muestran todos los logs centralizados mediante la interfaz web que provee Kibana.	Selecciona el componente "*Discover*". Selecciona el index "Logs". Selecciona los campos que desea visualizar.	Kibana mostrará los campos de los logs seleccionados	1- El usuario selecciona el componte "*Discover*" de Kibana. 2- El usuario selecciona los campos de los logs que desea ver.
No visualizar los logs.	No se muestran los logs o no se muestran uno o varios campos de logs.	Selecciona el componente "*Discover*". Selecciona el index "Logs". Selecciona los campos que desea visualizar.	Kibana mostrará el campo de log vacío.	

Tabla 12: Descripción de los escenarios del caso de prueba "Visualizar logs".

3.6 Resultados obtenidos de los casos de prueba.

Se utilizó el método de caja negra para realizar las pruebas sobre la interfaz del sistema. Donde se encontraron 6 no conformidades, en 3 iteraciones. De estas no conformidades 1 fue de error ortográfico y 5 de validación incorrecta. En la siguiente gráfica se muestra las iteraciones realizadas y las no conformidades detectadas, las resueltas y las pendientes (*Ver ilustración 11*).

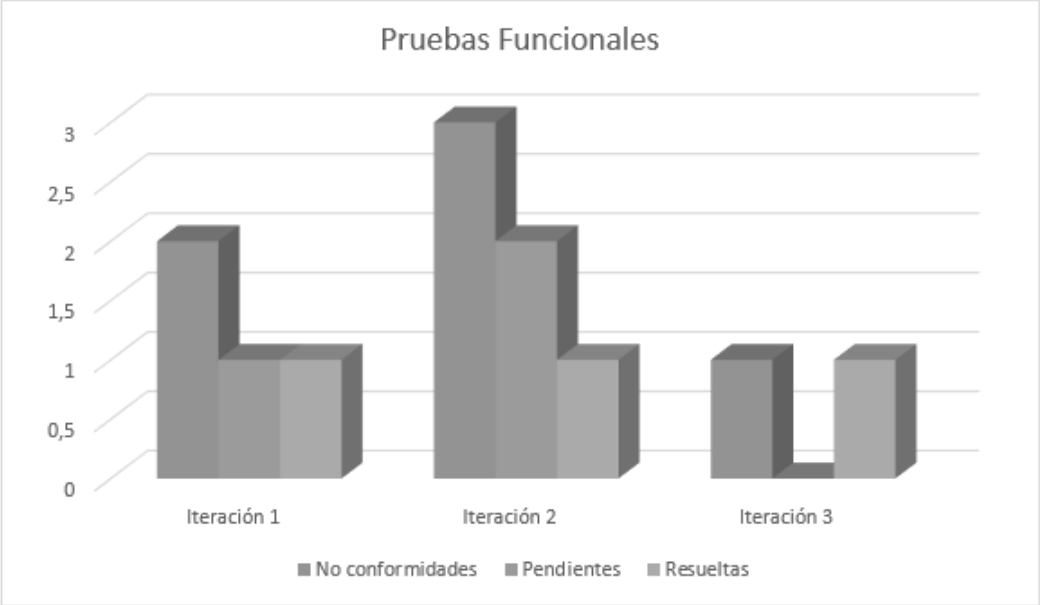


Ilustración 11: Estadísticas de las pruebas

Integración

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

Nivel 1	Componente	Descripción	Integración con el nivel superior
1	Módulo IDS	Módulo para la administración del IDS en Nova Servidores	Satisfactoria
2	Herramienta Nova_Server_Manager	Gestor de administración de nova servidores. Se encarga de la instalación, configuración y desinstalación de los servicios para servidores con que cuenta Nova Servidores 6.0.	

Tabla 13: Descripción de la integración del módulo IDS con la herramienta Nova_Server_Manager.

Resultado obtenido en las pruebas de integración

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

Conclusiones Parciales

En el desarrollo del capítulo se realizó el modelo de implementación del sistema para centralización y visualización de *logs* del IDS en Nova Servidores. A través del diagrama de despliegue se mostró como se distribuye la solución para su óptimo funcionamiento. Se especificó el uso de estándares de codificación para llegar a tener claridad y organización en el código fuente de la solución.

Se realizaron pruebas funcionales, unitarias, de integración y de aceptación para garantizar el correcto funcionamiento de sistema, resolviéndose así todas las no conformidades identificadas.

Conclusiones Generales

Al lograr el desarrollo de un sistema que permite la centralización y visualización de *logs* del IDS en Nova servidores se da cumplimiento al objetivo general planteado. Para llegar a ese resultado se cumple lo siguiente:

- El análisis y la fundamentación teórico-metodológico de los principales conceptos asociados a la investigación científica, permitió la conceptualización de los componentes de un sistema para la centralización y visualización de *logs* del IDS para Nova servidores lo cuales son los Snort desplegados en el sistema de red junto con barnyard2 y el servidor que contendrá la base de datos centralizada junto con el stack ELK.
- La aplicación de un modelo de madurez permitió elegir las herramientas adecuadas para el desarrollo del sistema para la visualización y centralización de *logs* del IDS para Nova Servidores.
- El uso de la arquitectura Modelo-Vista-Controlador unido al empleo de los patrones de diseño: alta cohesión y bajo acoplamiento permitió desarrollar el módulo IDS para la distribución de GNU/Linux Nova Servidores de manera que fue posible integrar su funcionamiento a la herramienta Nova_server_manager.
- La solución fue validada a partir de la ejecución de pruebas funcionales y de aceptación las cuales permitieron comprobar el correcto funcionamiento del sistema para la centralización y visualización de *logs* del IDS para Nova servidores.

Recomendaciones

Se recomienda:

- Empaquetar las diferentes herramientas utilizadas para el proceso de visualización de *logs* (Stack ELK) y subir al repositorio.
- Incorporar al repositorio de la universidad la herramienta Barnyard2.
- Incorporar más IDS para ampliar soporte del sistema desarrollado.

Referencias Bibliográficas

[1] REDACCIÓN JUDICIAL 2016. En 2015 aumentaron en 40% los ataques cibernéticos, dice la Policía. [en línea]. 22 marzo 2016. Disponible en: http://www.elespectador.com/noticias/judicial/2015-aumentaron-40-los-ataques-ciberneticos-dice-polici-articulo-623568.

[2] MONTES DE OCA, J.L. 2015. LA MIGRACIÓN HACIA SOFTWARE LIBRE EN CUBA: COMPLEJO CONJUNTO DE FACTORES SOCIALES Y TECNOLÓGICOS EN EL CAMINO DE LA SOBERANÍA NACIONAL. *Revista Universidad y Sociedad* [en línea], Disponible en: http://scielo.sld.cu/pdf/rus/v7n3/rus17315.pdf.

[3] GONZÁLEZ GÓMEZ DIEGO, 2003. *Sistemas de Detección de Intrusos*. julio 2003. S.l.: s.n.

[4] Pierra Fuentes, Allan. 2011.NOVA, DISTRIBUCIÓN CUBANA DE GNU/LINUX. REESTRUCTURACIÓN ESTRATÉGICA DE SU PROCESO DE DESARROLLO. La Habana: s.n., 2011. Trabajo final presentado en opción al título de Máster en Informática Aplicada.

[5] MICROSOFT CORPORATION. WINDOWS PROTOCOLS MASTER GLOSSARY. 2014.

[6] NIÑO MEJÍA DIANA CAROLINA y SIERRA MUNERA ALEJANDRO, 2007. *GUÍA METODOLÓGICA PARA LA GESTIÓN CENTRALIZADA DE REGISTRO DE EVENTOS DE SEGURIDAD EN PYMES*. ENERO 2007. S.l.: s.n.

[7] PEREZ PORTO JULIAN y GARDEY ANA 2014. Definición de. [en línea]. Disponible en: https://definicion.de/centralizacion/.

[8]`CABRERA JOSÉ MANUEL 2017. ¿Cómo y por qué centralizar tus logs? [en línea]. Disponible en: http://dbi.io/es/blog/caso-uso-centralizar-logs/.

[9] CHIRINO HORTA YEVGENI y DÍAZ CÁRDENAS DAYRENA, 2007. *Análisis y Configuración del Sistema de Detección de Intrusos Snort en la Universidad de las Ciencias Informáticas*. junio 2007. S.l.: s.n.

[10] QSOS CORE TEAM 2013. Collaborative technological watch. [en línea]. Disponible en:

http://www.qsos.org/.

[11] QSOS CORE TEAM, 2006. *Method for Qualication and Selection of Open Source software (QSOS)* [en línea]. abril 2006. S.l.: s.n. Disponible en: https://jose-manuel.me/thesis/references/qsos-1.6-en.pdf.

[12] CHHAJED SAURABH 2015. *Learnin ELK Stack*. Birmingham-Mumbai: s.n.

[13] GRAFANA TEAM 2017. Grafana Labs. [en línea]. Disponible en: http://staging-docs.grafana.org/.

[14]GRAFANA TEAM 2017. Grafana Labs. [en línea]. Disponible en: http://staging-docs.grafana.org/guides/basic_concepts/

[15] UPGUARD 2016. Top Free Network-Based Intrusion Detection Systems (IDS) for the Enterprise. & SCHREIBER JOE 2014. Open Source Intrusion Detection Tools: A Quick Overview. [en línea]. Disponible en:https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview.

[16] What is Snort? [en línea] 2017. Disponible en: https://snort.org/faq/what-is-snort.

[17] CLARK CHAMP 2014. Snort vs Suricata vs Sagan. [en línea]. Disponible en: https://github.com/Snorby/snorby/wiki/Snort-vs-Suricata-vs-Sagan.

[18] GEORGE KHALIL, 2015. *Open Source IDS high Performance Shootout* [en línea]. 2 febrero 2015. S.l.: s.n. Disponible en: https://www.sans.org/reading-room/whitepapers/intrusion/open-source-ids-high-performance-shootout-35772.

[19] ALBIN EUGIN, 2011. *A comparative analysis of the Snort and Suricata intrusion-detection systems.* [en línea]. septiembre 2011. S.l.: s.n. Disponible en: http://calhoun.nps.edu/bitstream/handle/10945/5480/11Sep_Albin.pdf?sequence=1.

[20] Suricata. [en línea] 2017. Disponible en: https://suricata-ids.org/.

[21] A Log File Extraction and Reporting System. [en línea] 2003. Disponible en: http://logrep.sourceforge.net/cgi-bin/logrep.

[22] JON MIDDLETON [sin fecha]. Logcheck(8) - Linux man page. [en línea]. Disponible en: https://linux.die.net/man/8/logcheck.

[23] Navarro, Juan Carlos Gómez. Servidor de Logs Centralizados. Valencia: Universidad Politécnica de Valencia, 2014.

[24] SWAYAM PRAKASHA 2016. Monitor Logs in Real-time with Swatch. [en línea]. Disponible en: https://opensourceforu.com/2016/08/swatch/.

[25] DAVID GULLETT, 2015. *Snort 2.9.7.2 and Snort Report 1.3.4 on Ubuntu 14.04 LTS Installation Guide*. de abril 2015. S.l.: s.n.

[26] BOOCH, G., et al.El lenguaje unificado de modelado. Addison-Wesley. 1999.

[27] GONZALEZ DUQUE RAÚL [sin fecha]. *Python para todos* [en línea]. S.l.: s.n. Disponible en: http://mundogeek.net/tutorial-python.

[28] JETBRAINS. JetBrains. [En línea] 2015. Disponible en: https://www.jetbrains.com/pycharm/.

[29] Oracle VM VirtualBox. [en línea][2017]. Disponible en: http://www.oracle.com/technetwork/es/server-storage/virtualbox/overview/index.html.

[30] Sommerville Ian (2006). Software Engineering, 8th ed. ISBN 0-321-31379-8. página 127 (en inglés).

[31] MIKE COHN 2008. *Agile Estimating and Planning* [en línea]. S.l.: s.n. Disponible en: http://synchronit.com/downloads/freebooks/Agile%20Estimating%20and%20Planning.pdf.

[32] D. Garlan y MARY SHAW 1993. *An introductioin to Software Arqitecture. Advances in Software Engineering and Knowledge Engineering* [en línea]. S.l.: s.n. Disponible en: https://books.google.com.cu/books?hl=en&lr=&id=k8bsCgAAQBAJ&oi=fnd&pg=PA1&dq=%22beyond+the+algorithms+and+data+structures+of+the+computation:%22+%22and+frameworks+for+systems+that+serve+the+needs+of+specific%22+%22as+industry+and+scientific%22+%22choices+among+design+alternatives.+Fourth,+an+architectural%22+&ots=m19z6kaXRh&redir_esc=y#v=onepage&q&f=false.

[33] PRESSMA R. 2009. *Software Engineering. A practitioner's Approach.* Séptima edición. Universidad de Connecticut: s.n. ISBN 978-607-15-0314-5.

[34] JONÁS A. MONTILVA y YAJAIRA RAMOS, [sin fecha]. *Patrones de Diseño para el Modelado de Redes en Sistemas de Información Geográfica*. S.l.: s.n.

[35] PRESSMAN R. 2002. *Ingeniería de Software, Un enfoque práctico.* Quinta edición. S.l.: s.n. ISBN 84-481-3214-9.

[36] FACULTAD DE INFORMÁTICA - UNIVERSIDAD POLITÉCNICA DE MADRID, 2002. *Patrones del «Gang of Four».* 2002. S.l.: s.n.

[37] POLO USAOLA MACARIO, 2015. *Patrones GRASP*. 2 febrero 2015. S.l.: s.n.

[38] CABEZA CHÁVEZ YANET 2015. *Módulo de JAVA para la herramienta Auditoría de Código Fuente.* La Habana: s.n.