SISTEMA MULTI-AGENTE DELIBERATIVO PARA LA OBTENCIÓN Y ANÁLISIS DE DATOS EN HONEYNETS

ERIK MICHEL GIRALDO GIRALDO

INGENIERO DE SISTEMAS Y TELECOMUNICACIONES

UNIVERSIDAD AUTÓNOMA DE MANIZALES

FACULTAD DE INGENIERIA

MAESTRÍA EN GESTION Y DESARROLLO DE PROYECTOS DE SOFTWARE

MANIZALES

2015

SISTEMA MULTI-AGENTE DELIBERATIVO PARA LA OBTENCIÓN Y ANÁLISIS DE DATOS EN HONEYNETS

ERIK MICHEL GIRALDO GIRALDO

INGENIERO DE SISTEMAS Y TELECOMUNICACIONES

Director de Tesis

GUSTAVO A. ISAZA ECHEVERRY, Ph.D.

UNIVERSIDAD AUTÓNOMA DE MANIZALES

FACULTAD DE INGENIERIA

MAESTRÍA EN GESTION Y DESARROLLO DE PROYECTOS DE SOFTWARE

MANIZALES

2015

Tabla de Contenido

Tabla de Figuras

Pág.

Lista de Tablas

Pág.

Resumen

Los sistemas de detección de intrusos (**IDS** – *Intrusion Detection System*) se basan en la utilización de firmas para la identificación de posibles intrusiones a áreas restringidas de organizaciones, las firmas que estos utilizan son generadas a partir del estudio de patrones de los atacantes. Dentro de los proyectos que aportan a la identificación de firmas se encuentra el proyecto *Honeynet*[1] en el cual se plantea una arquitectura orientada al análisis de atacantes en un único punto, es decir, solo existe un objeto que permite esta función y solamente ofrece una retroalimentación plana de los datos recolectados.

Dentro de las modificaciones a la arquitectura original de la *Honeynet*, se plantea el proyecto **GDH** - *Global Distributed Honeynet* en donde su arquitectura está ligada a la implementación de un *cluster*[2] de nodos, a través de una modificación específica sobre el sistema operativo Red Hat. La anterior propuesta implica una solución compleja a nivel de topología de Red.

Debido a la arquitectura original del proyecto *Honeynet*, los sistemas multi-agente nacen como una solución óptima para la mejora de esta, permitiendo el trabajo distribuido en pro de la recolección de datos para soportar la base de datos de firmas; adicional a esto, utilizando agentes deliberativos se aporta a la toma de decisiones con base a los datos capturados sin depender exclusivamente de una persona.

La propuesta del desarrollo de un sistema-multiagente deliberativo basado en la estructura estándar del proyecto *Honeynet* ofrece una mejora general a su arquitectura *stand-alone*[3] en la

[1] Es una arquitectura cuyo objetivo es plantear los componentes necesarios para exponer de manera consiente a atacantes, sistemas, aplicaciones y servicios reales y el acceso ilícito a estos; para la captura de comportamientos y formas de ataque.
[2] Conjunto de computadores construidos mediante la utilización de hardware o software y que se comportan como si fuesen un único computador.
[3] Elemento computacional que no hace parte de una red.

forma como los datos son extraídos y analizados, permitiendo la extracción de diferentes puntos (sin adiciones significativas a su núcleo) y la generación de conclusiones con base a esa recolección.

Con respecto al proyecto **GDH,** ofrece una solución menos acoplada e independiente de la implementación de cada *Honeynet*, ya que no es intrusiva en su topología y su arquitectura.

Palabras Claves: *Honeynet*, IDS, Agentes deliberativos, Sistema Multiagente

Abstract

The intrusion detection systems (**IDS** - Intrusion Detection System) is based on the use of signatures to identify possible intrusions into restricted areas of organizations, firms that they use are generated from the study of patterns of attackers. Among projects that contribute to the identification of signatures is the Honeynet project, which proposes an architecture oriented analysis for attackers in a single point, ie there is only one object that enables this feature and only provides feedback flat data collected.

Among the modifications to the original architecture of the Honeynet Project raises the **GDH** - Global Distributed Honeynet where architecture is linked to the implementation of a cluster of nodes through a specific modification of the Red Hat operating system. The previous proposal involves complex solution at Network Topology level.

Because the original architecture Honeynet Project, multi-agent systems are born as an optimal solution for the improvement of this, allowing the distributed work towards collecting data to support the database of signatures, in addition to this, using deliberative agents is provided to decision-making, based on the captured data without relying solely on a person.

The proposed development of a multi-agent system-based deliberative standard structure Honeynet Project provides an overall improvement to the architecture stand-alone in how the data is extracted and analyzed, allowing the extraction of different points (no additions Significant to the core) and generating conclusions based on this collection.

With regard to project **GDH,** this proposal provides an independent and less coupled implementation of the Honeynet, since it is not intrusive in its topology and architecture.

Keywords: *Honeynet,* IDS, deliberative agents, Multiagent System

INTRODUCCIÓN

El propósito de esta tesis es recolectar e interpretar los datos de ataques, identificados por una red *HoneyNet*, bajo un esquema distribuido y autónomo para el apoyo a la toma de decisiones, e identificación de comportamientos de ataques.

En la actualidad, los sistemas de misión crítica y de núcleo de negocio, son expuestos al mundo a través de internet, lo que los convierte en blanco de diferentes ataques para la obtención de acceso o de datos internos de cada organización. Dentro de las estrategias más comunes para la prevención y análisis de estos ataques, con miras a la obtención de patrones que permitan la generación de firmas únicas para la identificación de estos y poder ser contrarrestados de manera proactiva y reactiva, se puede encontrar entre otros, la utilización de software para la detección de intrusos o **IDS** (*Intrusion Detection System*), los cuales permiten la identificación de patrones y firmas en ataques que se estén realizando contra un sistema determinado.

Las firmas que utilizan los **IDS** son recolectadas con base a la identificación de patrones de ataque mediante la utilización de *Honeypots*[4] puros, de mediana interacción y de alta interacción.

Las *honeypots* de alta interacción, llamadas *Honeynets*, han sido concebidas para ser expuestas y comprometidas, lo que permite realizar un análisis y recolección de información de los ataques que se realicen, sin embargo, su estructura se pensó como un despliegue en un único componente, limitando la captura de información únicamente a la red en la cual están implementadas y sin una forma explícita de compartir los posibles patrones de ataques con otros componentes.

[4] Componente de Software cuyo objetivo es atraer atacantes mediante la simulación de elementos vulnerables, lo anterior con miras a la recolección de información.

Dentro de las posibles alternativas que permiten realizar un planteamiento para mejorar la arquitectura inicial que expone una *Honeynet* en búsqueda de eliminar las limitantes de la misma, mediante la adición de un comportamiento distribuido se pueden encontrar:

- Planteamiento de un sistema distribuido a través de **RPC**[5] (*Remote Procedure Call*)**, CORBA**[6] (*Common Object Request Broker Architecture*)**, RMI**[7] (*Remote Method Invocation*), esto permitiría la solicitud (ejecución remota) de funciones que consulten determinada información. El planteamiento de una arquitectura de este tipo implicaría el desarrollo, dependiendo de la solución escogida, del comportamiento, excluyendo de manera explícita la semántica necesaria para la comunicación y el desarrollo obligado de un protocolo de comunicación entre estos.
- Planteamiento de un sistema distribuido a través de *Web Service*[8] o *Restful*[9] (Representational State Transfer), esto permitiría la orquestación de los servicios necesarios para cumplir un objetivo, sin embargo, la forma de comunicarse está relacionada explícitamente al orquestador y al intercambio de mensajes entre los componentes de este. Adicional a lo anterior la orquestación está alineada en gran medida al modelamiento de procesos de negocio.
- Es importante clarificar y mencionar que una arquitectura distribuida en donde cada uno de los componentes cumple un objetivo y tiene una responsabilidad, es muy diferente en

[5] Protocolo que permite a un programa ejecutar código de manera remota en otro computador sin tener que preocuparse de manera detallada de la comunicación entre ambas máquinas. RPC fue un avance del uso tradicional de los Sockets

[6] Estándar que permite a diferentes componentes de software escrito en diferentes lenguajes de programación comunicarse.

[7] Es un método para invocar me manera remota métodos en diferentes computadoras, es exclusiva de Java y si se desea interactuar con otros lenguajes se deben de utilizar otras tecnologías.

[8] Tecnología que utiliza diferentes protocolos y estándares para intercambiar datos permitiendo una alta interoperabilidad mediante la adopción de estándares abiertos.

[9] Técnica de arquitectura de Software para sistemas distribuidos que se basa en la exposición de funcionalidades a partir de un identificador único y basándose en los diferentes métodos utilizados por el protocolo HTTP.

su concepto a un *cluster*, en donde el componente es replicado en diferentes nodos físicos y la carga o la alta disponibilidad se garantiza por el enrutamiento de las peticiones a cada uno de estos, haciéndolo de fondo una solución no pertinente al problema, no solo por su contexto sino por su objetivo.

- Implementar una arquitectura orientado a servicio bajo un modelo de arquitectura **ESB**[10] (*Enterprise Service Bus*) incluyendo diferentes componentes reutilizables, como por ejemplo servicios web.

En esta investigación se plantea una alternativa para lograr una mejora significativa en la arquitectura inicial, plasmando y logrando el modelamiento, diseño y desarrollo de un sistema multi-agente deliberativo que brinde la capacidad de recolectar y apoyar la deliberación sobre los ataques identificados. A partir de una instalación estándar de una red *HoneyNet*, se realiza el despliegue de un agente (*Extractor o Poller*) cuya responsabilidad es la extracción de los eventos de ataques capturados, para que esta información sea procesada y almacenada; una vez esto se logra por un agente (*Constructor o Builder*), la deliberación para realizar la detección de nuevos ataques es ejecutado un tercer agente, que utiliza razonamiento basado en casos y el algoritmo de clasificación **KNN**[11] (K-Nearest Neighbors) para encontrar los 5 casos que se consideren, acorde a una configuración base, posibles nuevos patrones.

De manera general la contribución de esta investigación se pude resumir de la siguiente manera:

[10] Modelo de arquitectura de software para realizar el diseño e implementación de esquemas de comunicaciones entre diferentes componentes, normalmente se usan servicios web como componente mínimo dentro de su implementación, aunque no se limita a estos.

[11] Algoritmo o método de clasificación supervisada que parte de los K ejemplos de entrenamiento para realizar una comparación base del nuevo escenario.

- Se revisa y define el modelo base sobre el cual se va a realizar la extracción inicial de ataques y los cuales se convertirán después de un procesamiento en la base de conocimiento inicial para la deliberación sobre nuevos patrones.
- Se realiza el diseño e implementación de los componentes o agentes asociados al proceso cooperativo de extracción, procesamiento, consulta, deliberación y almacenamiento de nuevos patrones de ataques.
- Se realiza el diseño e implementación del modelo de razonamiento basado en casos y los algoritmos de comparación necesarios para poder ejecutar un algoritmo de clasificación **KNN**, para encontrar los 5 casos que se pueden considerar nuevos patrones.
- Se realiza la ejecución del sistema y se obtienen resultados sobre la efectividad del mismo, obteniendo puntos de mejora y aserciones correctas en este.

Este documento se organiza de la siguiente forma. La primera parte presenta todo el referente contextual que consolida la base teórica del dominio del proyecto y los objetivos que se generan como resultado de ese análisis. La segunda parte presenta toda la estrategia metodología que acompaña todo el proceso gestión, diseño, implementación y pruebas de todo el proyecto. La tercera parte presenta el desarrollo del proyecto, iniciando con la definición del referente teórico hasta el análisis, diseño, implementación y pruebas concretas del sistema multi-agente que supone una solución a la carencia de un sistema realmente distribuido en una red *HoneyNet*.

Al finalizar el proceso, se puede identificar de manera clara un flujo de trabajo consistente de desarrollo involucrando cada una de las fases que se consideraron mínimas para garantizar la obtención del objetivo, apoyándose en un modelamiento orientado a agentes, en donde se puede identificar claramente los roles, la interacción entre los mismos, los comportamientos y las características que cada uno tiene. Los resultados obtenidos pueden evidenciar como el uso del

sistema multi-agente deliberativo sugiere nuevos patrones de ataque basado en una configuración previa, adicional, se pueden identificar puntos de mejora no solo en la implementación del mismo, sino en las funciones que soportan su funcionamiento y que pueden dar paso a la mejora del mismo.

1. REFERENTE CONTEXTUAL

1.1 Descripción del Área Problemática

Las empresas siempre han tratado de tener una visión holística de los elementos que comprometen o pueden llegar a comprometer el núcleo de su negocio, esto teniendo en cuenta que ya hace un buen tiempo las empresas están incorporando elementos como sistemas de misión crítica, sistemas de *e-commerce* y pagos en línea, sistemas de *servicedesk*, CRM[12] (Custom Relationship Management), según sea el caso, de manera pública o privada dentro de su cadena de valor, debido a las facilidades que estos elementos adicionan a su negocio y operación en el día a día.

El que las empresas coloquen todos estos elementos en Internet, es decir, de manera pública, implica que dentro de los riegos asociados a su negocio debe ser incluido la seguridad informática, práctica que no se tenía tan presente hace algunos años, pero que debido a los servicios actuales es necesario. Esto no solo se hace necesario cuando los servicios son prestados hacia el exterior de las organizaciones, sino que al interior de las mismas, el uso de técnicas como la Ingeniería Social[13] se está haciendo más común.

[12] Según expone (Buttle, 2009) "IT ha definido el CRM como la aplicación de software que permite la automatización de ventas, marketing y servicios del negocio" (Seguimiento a la relación con el cliente).
[13] Técnica o práctica utilizada para la obtención de información de personas mediante la manipulación de las mismas.

Uno de los proyectos y estándares a nivel del estudio de los diferentes patrones de ataque es el proyecto *HoneyNet* encargado de investigar los últimos ataques y desarrollar herramientas *Open Source* para mejorar la seguridad en Internet, este provee los lineamientos necesarios para comprender al atacante.

Debido a que cada día se están generando nuevas formas de ataque y con el fin de poder aprender de una forma más global, en masa y de forma distribuida los posibles escenarios de ataque[14] que esas nuevas formas conllevan, es conveniente el generar una forma de procesar la información capturada de una manera masiva y descentralizada que permita la generación de una base de datos de comportamientos y la generación de posibles conclusiones sobre estas.

El seguimiento a los nuevos tipos de eventos o ataques que se presentan en el día a día es complejo, ya que los sistemas que prestan la funcionalidad de detección de intrusos se basan en firmas, las cuales deben de ser actualizadas en la medida que existan alteraciones a los patrones y firmas. Dentro de su arquitectura tradicional, están concebidos para ser un sistema centralizado con un único punto de adquisición de datos, esto implica que:

- El límite en la captura de nuevos patrones estaría en un solo elemento, implicando un porcentaje realmente bajo de captura de ataques conocidos y desconocidos.
- Una única arquitectura implementada y por consiguiente un único objetivo con una sola estructura.
- Un único objetivo dentro de millones de máquinas reales y vulnerables.
- Acceso limitado a los datos recolectados.

[14] Conjunto de pasos que se siguen dentro de un contexto específico para realizar un ataque contra un sistema o red.

Si por el contrario se plantea una arquitectura distribuida el límite en la captura de información está limitado únicamente a donde los agentes no tengan acceso, ya que aun si se presentan diferentes plataformas, topologías y servicios implementados por diferentes personas, basados en los lineamientos del proyecto *HoneyNet*, la implementación de la solución se limita a la ejecución del agente y permisos relacionados a nivel de red, haciendo que la extracción y captura de información sea mucho más sencilla.

Debido a lo anterior y a que este tipo de sistemas tienen una gran carga de procesamiento de datos, el uso de sistemas distribuidos para la integración entre los diferentes **IDS** se hace notable.

Lo anterior permitiría una completitud mayor debido a que la posible actualización de firmas y deliberación sobre escenario de ataque no se hace con una única fuente de información (arquitectura original planteada en una *Honeynet*), sino que por el contrario, la información es extraída de diferentes fuentes, que en este caso, son despliegues nativos de una *Honeynet*, permitiendo que la actualización y el análisis de comportamiento sea mucho más rápido, aumentando la completitud en la identificación de patrones y vectores de ataque en el **IDS**.

1.2 Antecedentes

Diferentes investigaciones y trabajos se han venido desarrollados alrededor del uso de *HoneyPots* de mediana y alta interacción (*HoneyNets),* desde diferentes enfoques y áreas.

Una de las áreas en las que se ha venido trabajando es en la visualización de la información capturada dentro de una red comprometida, la cual es visualizada usualmente por elementos de monitoreo, (Becker, Eick, & Wilks, 1995) muestran como la visualización de patrones en el tráfico de una red es crucial en su monitoreo. Alonso et al. (2010) propone dentro de su investigación la utilización de modelos de proyección para visualizar los datos capturados dentro de una *HoneyNet*.

Döring (2005) dentro de su tesis de Maestría realiza un estudio de como el uso de *Honeypots* son valiosos dentro de la seguridad informática. Dentro de su documento realiza un recorrido por las definiciones más comunes, la planificación para su implementación y algunos casos de prueba, al igual que las conclusiones respectivas sobre su planteamiento.

Yin, Zhang, & Chen (2003) realizan la construcción de una *HoneyNet* con el único objetivo de recopilar datos, posteriormente y mediante la utilización de técnicas de minería de datos[15] obteniendo resultados de diferentes formas, como reglas de asociación de datos históricos o una estructura especifica utilizada por **IDS** o *Firewalls*.

Rammidi (2010) muestra el gran esfuerzo y arquitectura que el proyecto *HoneyNet* ha planteado, dando la seguridad que en su concepción de solución no distribuida su estrategia es lo suficientemente robusta. Dentro de sus conclusiones expone que el uso de *Honeypots* como complemento a la estrategias de seguridad.

Algunos proyectos externos se pueden encontrar:

- **GDH** - *Global Distributed Honeynet* en donde su arquitectura está ligada a la implementación de un *cluster* de nodos, a través de una modificación específica sobre el sistema operativo Red Hat.
- El proyecto *HoneyPot* que permite la instalación de un Software en un servidor Web y este automáticamente recoge estadísticas o el proyecto
- **DWH** – *Distributed Web HoneyPot* en donde se le entrega al usuario una máquina virtual a ser desplegada, cuya responsabilidad es la de actuar como un señor, capturar datos y realizar un análisis básico.

[15] Proceso computacional cuyo foco es el encontrar patrones en grandes cantidades de datos, esto, para procesarla y mostrarla en formato entendible con significado.

Al ser revisados, sin embargo, no dejan de ser utilidades tradicionales para la detección y prevención de intrusos con almacenamiento de datos, teniendo en cuenta, que estos son utilizados bajo entornos productivos y en el mejor de los casos se soportan en una *Honeynet* para exponer servicios vulnerables. Estos también carecen de autonomía, relegan la responsabilidad total del procesamiento al sensor, no tienen capacidad deliberativa sobre los diferentes ataques y su única función es la de recolectar y extraer estadísticas o tendencias sobre los datos capturados.

Khosravifar, Gomrokchi, & Bentahar (2009), hacen referencia al uso de un sistema multi-agente para la reducción de falsas alarmas dentro de un sistema de detección de intrusos, sin implicarlos directamente en el proceso de análisis y deliberación. Wang & Chen (2010), presentan como el uso de un sistema multi-agent y el aprovechar sus características de autonomía, ayudan de manera sustancial a aumentar el nivel de atención de posibles atacantes a una red *HoneyNet*, sin embargo, su foco es el aumento de atención, no la extracción y deliberación sobre los resultados obtenidos.

Herrero, & Corchado (2009), presentan un sistema multi-agente con la inclusión de un sistema de razonamiento basado en casos basado en creencias, deseos e intenciones para la detección y clasificación de ataques a partir de la técnica de detección anómala. Esta propuesta se basa en la detección de ataques, pero no en la identificación de nuevos patrones de ataque a partir de la identificación previa, ni tampoco en la recolección distribuida de eventos de posibles de ataques.

Kim & Kim (2012), basan su propuesta en que tanto los Sistemas de Detección de Intrusos (**IDS**) o Sistemas de Prevención de Intrusos (**IPS**[16]**)** no pueden identificar de manera inmediata ataques *0-day*[17], por lo cual, exponen la utilización de un sistema multi-agente que se encarga de

[16] Plataforma que ejerce el control sobre los accesos a un red con el objetivo de protegerla

[17] Ataque que se realiza contra un sistema o plataforma con base a vulnerabilidades encontradas que son desconocidas por las personas y fabricantes, por lo cual, aún no han sido solucionadas.

remover procesos maliciosos y archivos ejecutables de servidores, protegiendo la red de posibles ataques *0-day*.

En términos generales los proyectos e investigaciones que se han venido realizado focalizan sus objetivos en la identificación de ataques en diferentes momentos, componentes y capas, obteniendo o capturando paquetes a nivel de red para su procesamiento, pero no han atacado o buscado nuevos patrones de ataque dentro de los ya existentes, teniendo en cuenta que, un evento por sí mismo no necesariamente identifica un vector de ataque.

1.3 Justificación

Los sistemas de información y los servicios que estos prestan están expuestos día a día a diferentes tipos de incidentes de seguridad, esto debido a su naturaleza pública, a que se encuentran directamente conectados a la nube, a que la intranet donde se encuentran ha sido vulnerada o a que hay un atacante interno. Es por lo anterior que diferentes empresas han generado soluciones que atacan de una u otra forma este contexto y que aunque comparten algunos principios básicos cada uno tiene diferencias significativas en la forma de prever estos tipos de incidentes de seguridad. Cada vez que se presenta un incidente de seguridad se puede estar enfrentando a un nuevo patrón de ataque o a uno ya conocido, pero obtener la información sobre ese patrón es esencial para saber cómo protegernos, es por esto que la utilización de *HoneyNets* para la simulación de entornos vulnerables facilita el reconocimiento de los diferentes métodos de ataque.

El proceso de seguimiento a los ataques que se presentan dentro de un sistema o red, alineado a lo que las empresas actualmente han necesitado, dan como resultado el uso de Sistemas de Detección de Intrusos (**IDS**), basados en reglas y firmas que permiten la identificación de los posibles ataques, sin embargo, la generación constante de nuevos ataques y modificación de los mismos implica una actualización constante de firmas. Cada **IDS** es un ente único y como tal no

está pensado para interactuar con otros, lo que plantea un gran reto, esto debido a que cada **IDS** debe de procesar gran cantidad de información desde diferentes fuentes y la actualización de firmas y patrones se hace de manera individual (En un momento de tiempo todos podrían a llegar a tener las firmas actualizadas), pero está limitado por el ente que detecta y actualiza, si se trabajara en conjunto la actualización de firmas sería mucha más amplia y exacta, lo anterior se podría llegar a lograr con la utilización de Sistemas Multi-Agentes.

Debido a que los **IDS** están pensados para estar centralizados en una sola estación, gran limitante de los mismos, se han propuesto soluciones basadas en sistemas distribuidos.

Los principales problemas de utilizar arquitecturas tradicionales son Krmicek (citado por Isaza, 2010):

- La consola central tiende a ser un solo punto de fallo, en tal sentido, la red entera está sin protección si ésta falla.
- La escalabilidad es limitada. Procesar toda la información en un nodo implica un límite en el tamaño de la red que puede monitorizar.
- Se evidencian dificultades en reconfigurar o añadir capacidades y firmas a los **IDS.**
- El análisis de los datos de la red puede ser defectuoso, debido a su arquitectura centralizada, durante un periodo de fallo los datos capturados pueden llegar a ser defectuosos o presentar perdida de paquetes.

La integración de diferentes **IDS** y plantear un re-diseño de su arquitectura tradicional se hace evidente (Isaza, Castillo, & Segura, 2008), (Lips, & El-Kadhi, 2008), (Xin, Dickerson, & Dickerson, 2003), (Pan, Chen, Hu, & Zhan, 2003), por lo cual se plantea el uso de Sistemas Multi-Agente en conjunto con **IDS** para lograr la interacción entre diferentes elementos de este tipo.

El uso de Sistemas Multi-Agente como base fundamental del sistema distribuido permite que estos, a diferencias de arquitecturas distribuidas tradicionales, tengan un comportamiento basado en su entorno, basados en una semántica de comunicación y en sus características principales, en donde, todos trabajan para cumplir un objetivo, en este caso, extraer información de diferentes fuentes usando la misma semántica y deliberar sobre los datos obtenidos para la posible identificación de atacantes.

1.4 Formulación del Problema

El uso de *HoneyNets* es necesario para realizar un análisis correcto y preventivo de posibles patrones, herramientas y comportamientos dentro de un ataque a un sistema específico, sin embargo, el análisis que se realiza con las soluciones actuales centralizadas y no distribuidas, no tiene un componente deliberativo y su alcanzabilidad se limita a la identificación de amenaza con base a las firmas pre-cargadas, y al envío de notificaciones.

En la medida que se entiendan los posibles comportamientos y se identifiquen conocidos, para realizar una deliberación que apoye a los departamentos de seguridad informática, se podrían generar decisiones mucho más certeras y precisas. Con base a lo anterior se plantea la siguiente pregunta:

¿El uso de sistemas multi-agentes deliberativos en entornos distribuidos de *HoneyNets* (Despliegues diferentes bajo los lineamientos del proyecto *HoneyNet*) contribuye a procesos de razonamiento y descubrimiento de información anómala en entornos de monitoreo de seguridad?.

1.5 Objetivos

1.5.1 Objetivo General.

Diseñar e implementar un sistema multi-agente deliberativo que permita la recolección e interpretación de datos en *HoneyNets* distribuidos, que proporcione el análisis del comportamiento de los intrusos para toma de decisiones a partir de razonamientos deliberativos.

1.5.2 Objetivos Específicos.

- Modelar un sistema deliberativo basado en agentes, basado en sensores *HoneyNet* que cumpla con los estándares de esta iniciativa
- Analizar, diseñar, desarrollar e implementar un sistema multi-agente deliberativo y distribuido basado en **JADE**[18] (*Java Agent DEvelopment Framework*), que permita la obtención y análisis de datos de *HoneyNets*.
- Validar el comportamiento deliberativo de los diferentes vectores de ataque frente a patrones y vectores ya conocidos.

[18] Marco de desarrollo, implementado en Java para el desarrollo de sistemas multi-agente a través de un componente de capa media que cumple con las especificaciones FIPA

1.6 Resultados Esperados

El aporte de este proyecto está enfocado directamente en la generación de un sistema que permita la recolección de datos de diferentes *HoneyNets* ubicadas geográficamente en lugares diferentes. La generación del sistema esta soportada en un sistema multi-agente que involucre el censo de las diferentes *HoneyNets* y el análisis de la diferente información.

Dentro del aporte se puede incluir la generación de un modelo de base de datos lo menos relacional posible para que permita la aplicación posterior de diferentes técnicas de extracción de información o la extracción de patrones mediante inferencias, sin preocuparse por relaciones complicadas o limitaciones a nivel de claves foráneas.

El aporte a la comunidad es la generación de una base de información que contenga de manera clara las lecciones aprendidas de éxito y de fracaso durante todo el proceso, desde la generación de la base de datos para almacenar la información de los diferentes ataques, el desarrollo del sistema y en si el proceso en general.

- **Fortalecimiento de la comunidad científica.**

Tabla 1 *Fortalecimiento de la comunidad científica*

Resultado/Pro ducto Esperado	**Indicador**	**Beneficiario**
Prototipo de Sistema Multi-Agente para red de sensores honeynets		Comunidades especializadas

- **Apropiación social del conocimiento.**

Tabla 2 *Apropiación social del conocimiento.*

Resultado/Producto Esperado	Indicador	Beneficiario
Difusión a la comunidad Científica	Artículo	Comunidad Académica
Informe final de proyecto de Investigación	Archivo digital y disponible en la biblioteca de la UAM	Comunidad Académica

2. ESTRATEGIA METODOLÓGICA

2.1 Metodología

2.1.1 PSP.

Debido al carácter del proyecto el proceso de desarrollo a utilizar es **PSP**[19] **0,** que permite de manera individual un seguimiento al proceso de desarrollo, determinando entregables básicos, métricas y elementos de seguimiento.

En la figura 1 Humphrey (2000) muestra las diferentes etapas, sus entradas y sus salidas.

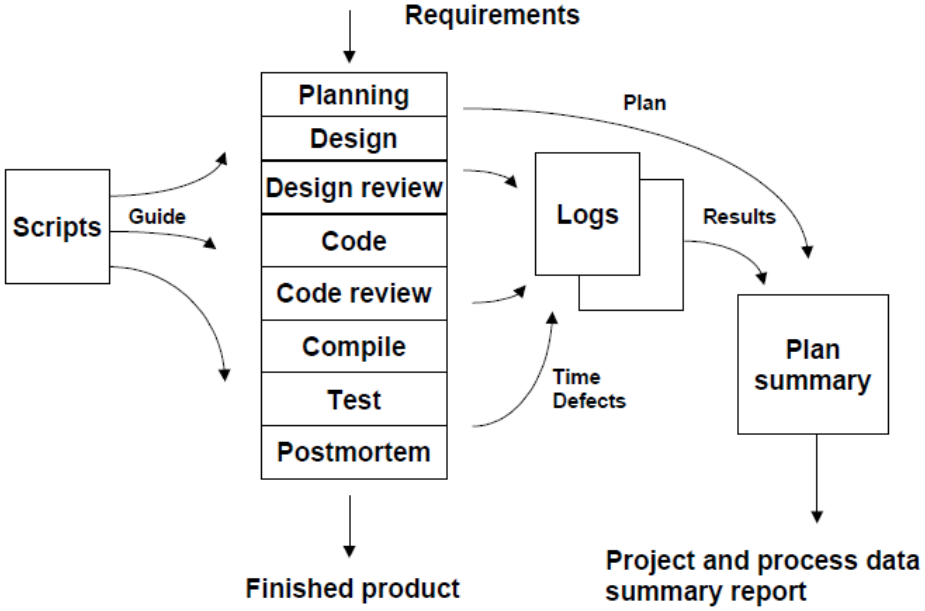

Figura 1. Flujo de Procesos de PSP0. (Humphrey, 2000)

Debido a que el proyecto implica el entendimiento de conceptos no claros en un principio y su carácter iterativo, se decide utilizar PSP0 en su modo cíclico:

[19] Proceso para la construcción de software, formado por métodos, formularios y scripts que le muestran a los ingenieros como planear, medir y gestionar su trabajo.

En la figura 2 (Carneige, 2006) Presenta las etapas, entradas y salidas de proyectos extensos y para los cuales son necesarios dividir en submódulos.

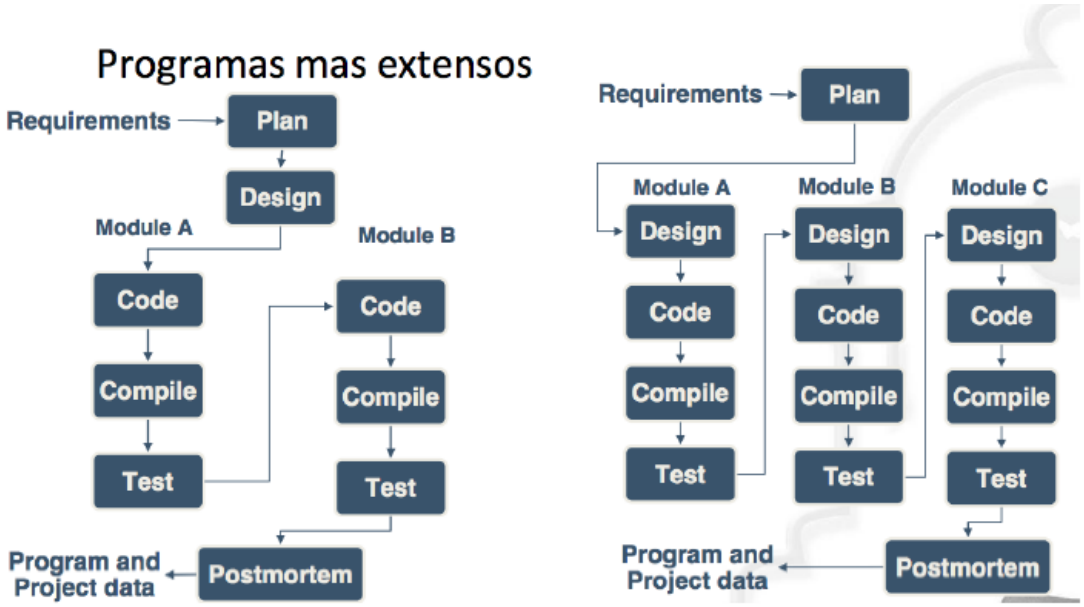

Figura 2. Flujo de Procesos de PSP0, para programas más extensos. (Carneige, 2006)

Elementos del proceso:

1. **Plan:** Producir un plan para desarrollar el programa definido en los requerimientos.
2. ***Design*:** Producir una especificación del diseño del programa.
3. ***Code*:** Transformar el diseño en instrucciones de lenguaje de programación.
4. ***Compile*:** Trasladar las instrucciones en código fuente al código ejecutable.
5. ***Test*:** Verificar que el código ejecutable satisfaga los requerimientos.
6. ***Postmortem*:** Resumir y analizar los datos del proyecto.

2.1.2 UPSAM.

El grupo de Investigación en Agentes de Software de la Universidad Pontificia de Salamanca en Madrid ha desarrollado dentro de uno de sus proyectos la metodología **UPSAM**[20], enfocada en la facilitación de la construcción de sistemas de agentes de complejidad media.

UPSAM enmarca el proceso de análisis de Sistemas multi-agentes en el paradigma del modelado de agentes y permite alinearlo al proceso de diseño de **RUP**[21].

(Castillo, 2004) dentro de su propuesta hace uso de la metodología para la realización de un análisis y diseño organizado de un sistema orientado a Agentes. (Castillo, 2004) contextualiza la metodología en el uso algunos procesos de **RUP** (Proceso Unificado de Rational) orientados al desarrollo de agentes, dirigido por casos de uso y centralizado en la arquitectura. (Castillo, 2004) muestra los modelos asociados a **UPSAM,** de los cuales se extraen:

- Modelo de Tareas: Servicios, tareas, objetivos y funcionalidades asociados a roles, este modelo normalmente se expresa a través del uso de casos de uso orientado a agentes.
- Modelo de Arquitectura de la organización de agentes: Descripción de agrupación de los agentes y sus relaciones, en este es normal la utilización de un diagrama general de actividades acompañado de la especificación de responsabilidades, propósitos, percepciones, tareas y salidas.
- Modelo de Agentes: Descripción de agentes a instancias, comportamiento y como realizar el control especificado, en este, es común la utilización de diagramas de clases basado en los conceptos claves de los roles asociados a los agentes.

[20] Lenguaje de modelamiento UML, focalizado a agentes inteligentes.
[21] Proceso de desarrollo de Software que provee un conjunto de pasos y/o metodologías aplicables y adaptables a las necesidades de cada organización y/o proyecto.

- Modelo de Comunicación de Agentes: Descripción de la secuencia de interacción entre los diferentes tipos o roles, este, es normalmente especificado a través de diagramas de secuencia.

En la Figura 3 (Castillo, 2004) categoriza cada una de las etapas asociadas según sea la fase en la que se extraen los elementos a utilizar y se encuentre de la siguiente manera:

Metodología	Vista o Fase de Análisis	Vista o Fase de Diseño
UPSAM	Modelo de Tareas Modelo de Arquitectura de	Modelo de Agentes Modelo de Comunicación de Agentes

Figura 3. Fases UPSAM. (Castillo, 2004)

2.2 Pruebas

Las pruebas se enfocarán en la validación de los agentes deliberativos. Dentro del esquema planteado la capacidad de deliberar de los agentes se logrará a través de la implementación de **CBR**[22] (*Case Based Reasoning*), cuyo flujo puede ser visualizado en la siguiente figura:

En la figura 4 Pinzón, Herrero, De Paz, Corchado, & Bajo (2010) muestra las diferentes etapas aplicadas dentro del flujo de ejecución de un CBR.

[22] Técnica de inteligencia artifikrmicial para soportar la toma de decisiones y el aprendizaje en sistemas de decisión complejos.

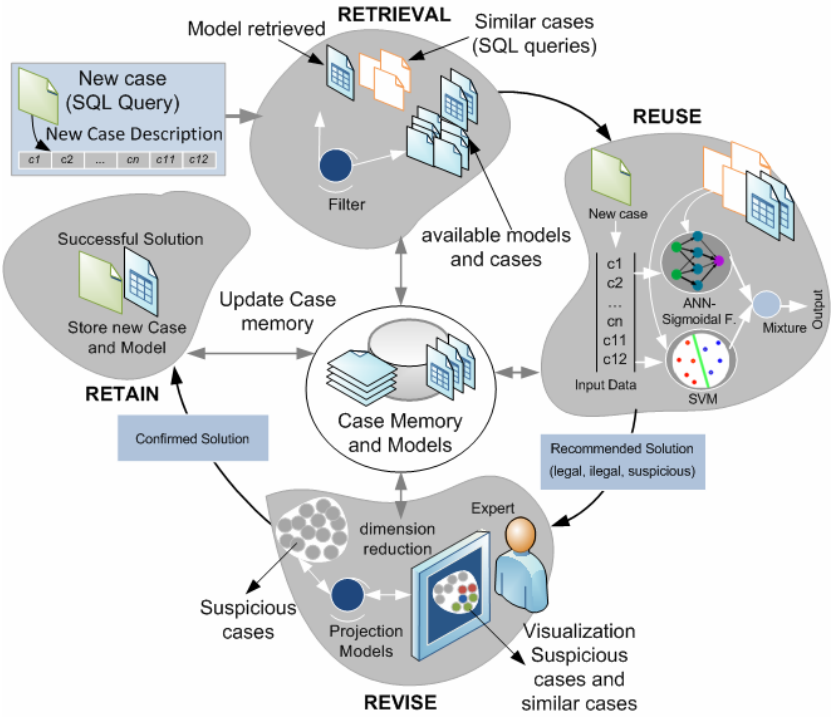

Figura 4. Clasificación y ciclo de un sistema CBR. Pinzón, Herrero, De Paz, Corchado, & Bajo (2010)

Como se puede visualizar, el flujo permite de manera inherente la validación de los casos nuevos casos que se presenten, mantenerlos en caso de ser correctos para alimentar la base de conocimiento y desecharlos en caso de que no.

Las pruebas consistirán en la validación de las deliberaciones que se hagan, el análisis de un experto o de casos previos de estos y su almacenamiento en caso de ser correctos, esto se hará en como mínimo dos nodos de *HoneyWalls*[23].

Adicional a lo anterior se van a realizar pruebas de Sistema y de Integración para garantizar que el flujo completo y el sistema en general sean funcionales, y realicen las operaciones de manera correcta.

[23] Componente intermedio encargado de filtrar, observar y procesar el tráfico que se genera hacia una o varias HoneyPots

3. DESARROLLO

3.1 Referente Teórico

3.1.1 Seguridad Informática.

La seguridad informática busca un equilibro entre los principios de integridad (la información no puede ser alterada o modificada como consecuencia de incidentes o accesos indebidos), la confidencialidad (la información solo puede ser accedida por usuarios autorizados) y la disponibilidad (el sistema debe operar normalmente en el momento en que sea requerido) en un ambiente informático (Kumar, 1995).

3.1.2 Honeypot.

Un *HoneyPot* es un elemento o componente que representa parte o la totalidad de un sistema de información, en donde, su valor es el uso ilícito de recursos. La interacción de entidades externas implica de manera explícita una interacción maliciosa con el componente. Un *HoneyPot* puede ser categorizado de la siguiente manera:

- De baja interacción: Su concepción base es a simulación de servicios y sistemas operativos. Se basan en la interacción a partir de escenarios pre-definidos dentro del *HoneyPot*, los cuales se forman a partir de eventos y respuestas.
- De alta interacción: Se presentan a través de una arquitectura definida compuesta por toda una red de computadores diseñados para ser atacados. El objetivo principal es la creación de un ambiente controlado, en donde toda actividad es capturada y procesada. El ejemplo más tangible de esto es una *HoneyNet*.

3.1.3 HoneyNet.

Una *Honeynet* representa un tipo de *Honeypot*, especialmente, de interacción alta y concebida con el objetivo de capturar alta cantidad de datos sobre posibles amenazas. Una *HoneyNet* provee sistemas, aplicaciones y servicios reales para que los atacantes puedan interactuar, todos estos corriendo bajo un entorno altamente controlado.

Las *HoneyNets* se consideran arquitecturas definidas para la recolección de datos, en las cuales los elementos claves a identificar son:

- *HoneyWall*: Es un intermediario que separa los *HoneyPots* del resto de elementos y del exterior. Todo el tráfico dirigido o de salida de cualquier *HoneyPot* debe de pasar por este, siendo un elemento transparente e invisible a cualquier objeto interactuando con los *HoneyPots*. HoneyNet Project (2006) refiere que un *HoneyWall* debe de implementar al menos las siguientes características:
    - Control de Datos: Se basa en que se le permite al atacante realizar, es decir, entre más acciones se le permita a un atacante ejecutar dentro de una *HoneyNet* más se aprenderá de él, pero, entre más esto sea así, más oportunidades tendrá de utilizar la *HoneyNet* de manera maliciosa. Es debido a lo anterior, que se debe de mantener un equilibrio en la libertad que se le sede al atacante.
    - Captura de datos: Debe de permitir la captura y almacenamiento de la mayor cantidad de información sobre las acciones realizadas por el atacante, sin que este monitoreo sea detectado.
    - Análisis de datos: Debe de permitir el análisis de los datos recolectados previamente. Una *Honeynet* sin un elemento que permita realizar análisis no es de mucha utilidad.

En la figura 5 (HoneyNet Project, 2006) muestra los diferentes elementos de una Arquitectura *HoneyNet*.

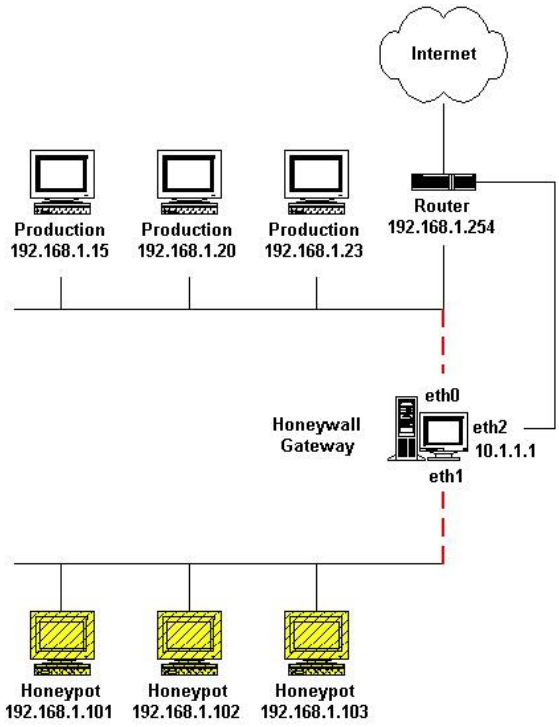

Figura 5. Arquitectura base de una HoneyNet. (HoneyNet Project, 2006)

3.1.4 Sensor HoneyNet.

Elemento que conforma una arquitectura distribuida de *HoneynNets*, encargado de la captura de gran cantidad de datos que se dirijan a uno de los *HoneyPots* que la conforman para su posterior uso.

3.1.5 Proyecto HoneyNet.

El proyecto *HoneyNet* nace como una organización sin ánimo de lucro conformada por especialistas de la seguridad informática dedicados la investigación de amenazas informáticas mediante el despliegue de diferentes redes en todo el mundo para sean atacadas de manera controlada. Su misión es clara al tratar de conocer las herramientas, tácticas y motivos de la comunidad *blackhat*[24], y compartir las lecciones aprendidas.

El proyecto define tres objetivos principales:

- Incrementar el conocimiento de las amenazas existentes.
- Educar e informar sobre las amenazas.
- Dar la capacidad a las organizaciones de conocer más por su cuenta.

3.1.6 Agente.

Tecuci (1998) define un agente de la siguiente forma:

> Un agente inteligente es un sistema basado en conocimiento que percibe su entorno (el cual puede ser el mundo físico, un usuario a través de una interfaz gráfica de usuario, un grupo de otros agentes, la Internet, u otros ambientes complejos); razonan para interpretar percepciones, infieren, resuelven problemas, definen acciones; actúan sobre el entorno para materializar un conjunto de objetivos o tareas para las cuales fue diseñado. El agente interactúa con un humano o algún otro agente a través de algún tipo de lenguaje de comunicación de agentes y no obedece ciegamente, pero puede tener la habilidad de modificar requerimientos, responder preguntas de aclaración, o incluso rehusarse a

[24] Persona con un alto conocimiento técnico cuyo objetivo es violar la seguridad asociada a un componente informativo con un fin personal o malicioso.

satisfacer ciertas peticiones. Un agente puede aceptar solicitudes de alto nivel de acuerdo a lo que quiera el usuario y puede decidir cómo satisfacer cada solicitud con algún grado de independencia o autonomía, exhibiendo comportamiento orientado a objetivos y escogiendo dinámicamente que acciones llevar a cabo, así como en que secuencia. Puede colaborar con el usuario para mejorar el cumplimiento de sus tareas o puede tomar estas tareas en beneficio del usuario, y para hacer esto emplea algún conocimiento o representación de los objetivos o deseos del usuario. Puede monitorear eventos o procedimientos para el usuario, puede aconsejar al usuario sobre como ejecutar una tarea, puede entrenar o enseñar al usuario, o puede ayudar a diferentes usuarios. (p.1)

Otra definición interesante la hace FIPA[25] (2003) diciendo: *"*Un agente es un proceso computacional que implementa la autonomía y la funcionalidad de comunicación de una aplicación*"*.

3.1.7 Sistema Multi-Agente.

Castillo (2004) define un Sistema Multi-Agente como "un grupo de agentes que interactúan entre sí para conseguir objetivos comunes".

> Un Sistema Multi-Agente agrupa elementos como el entorno, un conjunto de objetos (que se integran con el entorno que pueden ser percibidos, creados, modificados y destruidos por diferentes agentes), agentes (representan las entidades activas), relaciones entre objetos y/o agentes y operaciones para que se generen percepciones, producciones, consumos, transformaciones y manipulaciones de objetos. Isaza (Citado por Ferber, 1999)

Según Isaza (citado por Sanjuan, 2006) los agentes pueden clasificarse con base en diferentes criterios:

[25] (Foundation for Intelligent Physical Agents) Fundación encargada de la definición del estándar asociado al comportamiento de agente.

- Según su movilidad pueden ser móviles (se pueden encontrar en distintos estados de tiempo en distintos dispositivos) o estáticos (siempre mantienen su dispositivo base).
- Por su comportamiento pueden ser Deliberativos (donde se procesan modelos de razonamiento simbólicos permitiendo planificar sus acciones y negociaciones con otros agentes) y Reactivos (No cuentan con modelos simbólicos de razonamiento, actúan en consecuencia a estímulos).
- Por sus características como colaborativos (trabajan mancomunadamente para lograr un objetivo), de interfaz (facilitan la comunicación con los usuarios), móviles (pueden moverse por el entorno), de información (recuperan información a partir de diferentes fuentes), reactivos (se comportan como reacción a estímulos), híbridos (se asocian a diferentes categorías), ideales (cumplen las categorías principales), emocionales (son agentes que poseen estados emocionales y que interactúan con el mundo externo para recibir entradas y enviar salidas posiblemente en tiempo real) Isaza (citado por Camurri, & Coglio, 1998).

3.1.8 Agentes Deliberativos.

Isaza (citado por Bratman, 1987) define a los agentes deliberativos como aquellos que utilizan un modelo simbólico para representar su conocimiento, parte de la base filosófica de la teoría de las Creencias-Deseos-Intenciones presentada por Bratman, como modelo de razonamiento práctico humano. En el modelo BDI la estructura interna de un agente y su capacidad de elección se basa en aptitudes mentales. Esto tiene la ventaja de utilizar un modelo natural (humano) y de alto nivel de abstracción. El modelo **BDI** utiliza "*Beliefs*" como aptitudes informativas, "*Desires*" como aptitudes motivacionales e "*Intentions*" como aptitudes deliberativas de los agentes. (Isaza, 2010, 62)

3.1.9 Lenguajes de Agentes y Multi-Agentes.

(Castillo, 2004) dentro de su investigación hace un seguimiento detallado a la evolución de los lenguajes utilizados por el desarrollo y comunicación de agentes y multi-agentes. Esta fue de gran importancia para este estudio al igual que las referencias y documentos utilizados por (Isaza, 2010). Con base a lo anterior (Isaza, 2010) menciona los siguientes lenguajes:

- *KQML (Knowledge Query and Manipulation Lenguaje*): Ofrece una infraestructura para enviar mensajes entre agentes, es un lenguaje independiente de la sintaxis, el contenido, de las ontologías y de los protocolos de transporte (TCP, IIOP, SMTP, entre otros).
- *ACL-FIPA:* Esta organización propuso un lenguaje estándar para la comunicación entre agentes conocido como *ACL-Agent Communication Language* (Lenguaje de Comunicación entre Agentes) (Fipa, 2003). El principal objetivo es darle un sentido semántico a los mensajes que se cruzan entre agentes; su base son los actos comunicativos a partir de la acción de los agentes. En el siguiente diagrama se puede visualizar la interacción de un mensaje FIPA-ACL; la estructura de este tipo de mensajes está conformada por diferentes parámetros requeridos para la comunicación entre agentes.

3.1.10 Jade (Java Agent Development Framework).

Isaza (2010) define: "Jade es un *middleware*[26] que ofrece tanto un ambiente de desarrollo como de ejecución para la realización de aplicaciones distribuidas multi-agentes basadas en comunicación".

[26] Software intermedio que permite la comunicación con otras capas, normalmente entre la capa de aplicación e inferiores.

Isaza (citado por Bellifemine, Caire, & Greenwod, 2007) Este entorno evoluciona de manera dinámica con sus actores o agentes, que aparecen y desaparecen del sistema de acuerdo con las necesidades y requisitos de la aplicación. La comunicación entre los agentes es simétrica asumiendo su correspondiente rol de origen y destino.

JADE esta implementado en *Java* y ofrece un conjunto de utilidades que facilitan el control, administración de los agentes, basados en principios de:

- *Interoperabilidad:* JADE cumple con las especificaciones FIPA, garantizando que los agentes interactúen con otros agentes que poseen el mismo estándar así no estén implementados con JADE.
- *Portabilidad y Uniformidad:* JADE ofrece un conjunto de librerías independientes de los recursos de la red y de la JVM[27] (*Java Virtual Machine*).
- *Intuitivo:* La complejidad está encapsulada en un middleware reflejado en una API[28] (Application Programming Interface) simple y administrable. (Isaza, 2010)

3.1.11 PSP (Personal Software Process).

Actualmente la ingeniería de Software es una de las más grandes e influyentes industrias, sin embargo, ha recibido grandes griticas por la baja calidad en los productos que son desarrollados.

PSP (*Personal Software Process*) ha surgido como una metodología formal y estructurada para el desarrollo de Software, que mediante una definición clara de procesos permite a ingenieros producir productos y servicios de alta calidad dentro de un alcance y un presupuesto

[27] Componente base de la plataforma java que se encarga de ejecutar una aplicación en una arquitectura específica
[28] Conjunto de rutinas específicas que exponen funcionalidades asociadas a una plataforma o componente

definido. Esta metodología define de manera global tres fases genéricas (Poweroy-Huff, Cannon, Chick, Mullaney & Nichols, 2009):

- Planeación: Producir un plan para hacer el trabajo
- Desarrollo: Realizar el trabajo
    - Definir los requerimientos
    - Diseñar el programa
    - Revisar el diseño y corregir los defectos
    - Codificar
    - Revisar el código y corregir los defectos
    - Construir o compilar y corregir los defectos
    - Probar el programa y corregir los defectos
- Postmortem: Comparar el rendimiento real con el planificado, almacenar datos procesados, producir en resumen y documentar todas las ideas para mejoras.

PSP está compuesto por 4 componentes:

- Scripts: Descripciones de experto que hacen referencia a formularios, estándares, checklists, sub-scripts y medidas. Este se encarga de documentar:
    - El objetivo del proceso
    - Criterios de entrada
    - Recomendaciones generales
    - Fases a ser realizadas
    - Medidas y pasos a realizar
    - Condiciones de salida

- Formularios: Promueven un marco consistente para la recolección y almacenamiento de datos, especificando que datos son necesarios y donde almacenarlos.
- Medidas: Cuantifican el proceso y el producto.
- Estándares: Proveen definiciones precisas y consistentes que guía el trabajo y la recolección y uso de los datos.

PSP ha mostrado una evolución y un incremento en su definición de procesos y madurez, dentro de lo cual se pueden definir 3 niveles base subdivididos en 2 cada uno.

- PSP0: En el cual se deben de llevar los siguientes registros
  - Proceso actual
  - Registro de tiempos
  - Registro de defectos
  - Estándar de defectos

- PSP0.1: Se adicionan a los elementos de PSP0:
  - Estándares de codificación
  - Medidas de tamaño
  - Propuesta de mejoramiento

- PSP1:Se adicionan a los elementos de PSP0.1:
  - Estimación de tamaño
  - Reporte de pruebas

- PSP1.1: Se adicionan a los elementos de PSP1
  - Planificación de tareas
  - Planificación de Cronograma
  - Revisión de código

- PSP2: Se adicionan a los elementos de PSP1.1
  - Revisión de código
  - Revisión de diseño

- PSP2.1: Se adicionan a los elementos de PSP2
  - Planificación de diseño

3.2 Desarrollo del Proyecto

A continuación se presenta en una tabla de especificación de requerimientos, un resumen del alcance del proyecto, el cual, también se complementa con los formatos asociados a cada iteración incremental del proceso PSP y que se pueden encontrar como anexos (ver).

3.2.1 Fase de Planeación.

Tabla 3 *Especificación de requerimientos del proyecto.*

Rol	Necesidad	Objetivo	Criterios de aceptación
Usuario	Obtener los posibles ataques que se detectan en la red	Generar y alimentar la base de casos	1. Consulta SQL eficiente 2. SQL preciso al esquema utilizado

			por SNORT[29] 3. SQL que permita realizar la consulta asociada
Usuario	Identificar de cada ataque la IP origen, la IP destino, el evento asociado y la ocurrencia	Generar y alimentar la base de casos	1. Consulta SQL eficiente 2. SQL preciso al esquema utilizado por SNORT 3. SQL que permita realizar la consulta asociada
Usuario	Tener una aplicación liviana, desacoplada de la arquitectura del SO que permita a consultar los ataques y obtener los campos especificados	Garantizar de manera masiva su implementación para la obtención de vectores de ataque geográficamente distribuidos	1. Aplicación Java 2. Ejecución de la aplicación con una máquina virtual estándar en un sistema operativo Windows 3. Ejecución de la aplicación y visualizar la consulta de datos 4. Ejecución de la aplicación y búsqueda de servicio de construcción de casos
Usuario	Procesar los datos capturados y convertirlos en un formato de texto fácil de enviar	Estar alineado con la especificación FIPA.	1. Vector de ataque en formato JSON
Usuario	Tener una aplicación liviana, desacoplada de la arquitectura del SO que permita recibir datos, procesarlos y alimentar la base de conocimiento. Esta debe de estar en la capacidad de determinar su capacidad y responder acorde a esto.	Ofrecer una manera de sencilla de construir casos a partir de vectores de ataque para alimentar la base de datos de conocimiento, adicional, de ofrecer una manera desacoplada y transparente en su ubicación de procesar estos datos. Lo anterior garantiza un despliegue masivo y disperso geográficamente.	1. Aplicación Java 2. Ejecución de la aplicación con una máquina virtual estándar en un sistema operativo Windows 3. Ejecución de la aplicación, recepción de petición de construcción y visualización de almacenamiento de datos como casos 4. Aplicación registrada en las páginas amarillas 5. Aplicación descubrirle por otros componentes
Usuario	Tener un punto centralizado donde se publiquen y ofrezcan servicios tanto de consulta como de construcción de casos y vectores de ataques	Garantizar las implementaciones de unas páginas amarillas donde se puedan registrar, publicar y consumir servicios de diferentes componentes, garantizando transparencia en la ubicación geográfica de estos.	1. Aplicación Java 2. Aplicación que permita el registro y consulta de servicios expuestos por componentes

[29] Sistema de detección de intrusos.

Usuario	Configurar de manera predeterminada los campos IP origen, IP destino y ataques asociados (Evento, Fecha y Ocurrencia) con peso 1 para realizar consultas de KNN.	Dejar por defecto una configuración base para ejecutar la consulta de apoyo al CBR.	1. Visualización de configuración base con estos campos 2. Funciones de similitud implementadas en Java
Usuario	Tener una interfaz que me permita configurar un nuevo patrón y buscar sobre la base de conocimiento inicial los casos aproximados.	Facilitar la búsqueda de nuevos patrones dentro de los ya configurados ya actualizados.	1. Aplicación Java 2. GUI que permita la configuración de los campos especificados 3. Llenado automático de campos a partir de la base
Usuario	Poder instalar o desplegar de manera sencilla diferentes aplicaciones, tanto de recolección como de construcción, que puedan interactuar entre si y que permitan publicar sus servicio en un punto centralizado	Fácil distribución	1. Registro de al menos un componente de cada uno en las páginas amarillas
Usuario	Consultar los 5 primeros casos cuyos criterios se adecuen a los configurados y persistirlos en base de datos para alimentar la base de conocimiento	Facilitar la implementación y dejar por defecto una configuración base útil para revisar la propuesta.	1. Visualización de consultas realizadas 2. Nuevos casos almacenados en base de datos
Usuario	Permitir y dejar abierto la opción de implementar la revisión de los casos y evitar el almacenamiento de manera automática	Garantizar la inclusión de la etapa de revisión dentro del flujo de CBR.	1. Identificación clara del lugar para modificar los criterio 2. Mostrar y/o especificar la forma de implementar
Usuario	Realizar ataques controlados dentro de un laboratorio y entornos vulnerables.	Poder simular ataques y vectores de ataque para que sean almacenados y generar una base confiable.	1. Realización de ataque desde una maquina a otra 2. Verificación y seguimiento al ataque 3. Visualización en base de datos

3.2.2 Fase de Desarrollo.

3.2.2.1 Análisis y diseño del Modelo.

Las características asociadas de autonomía, reactividad, proactividad, movilidad, inteligencia, entre otras, sugieren y sustentan la necesidad de dividir de manera sistémica los

diferentes actores que interactúan con las diferentes capacidades asociadas a los agentes. Lo anterior soporta la interacción compleja entre los diferentes agentes para lograr un objetivo común.

En la figura siguiente 6, se presentan los diferentes modelos de casos de uso que intervendrán dentro el sistema diseñado.

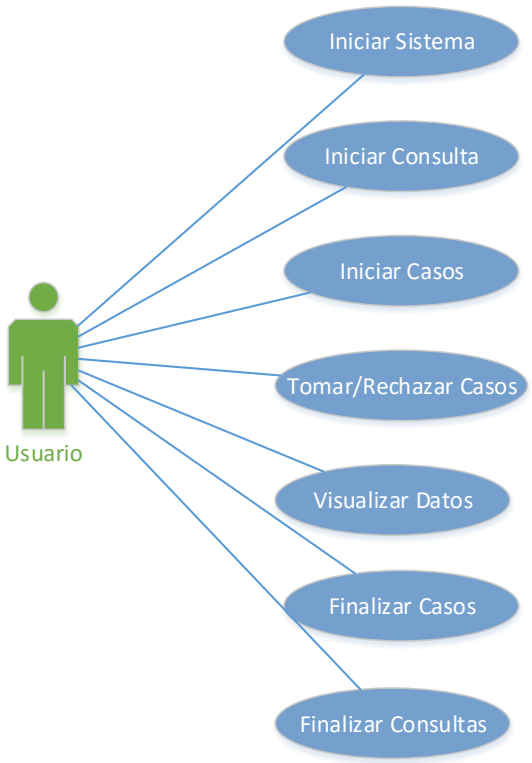

Figura 6. Modelos de Casos de Uso General en el sistema

En este modelo se presenta la iteración del usuario (rol Experto) con el sistema.

Los escenarios posibles para este usuario son:

- Iniciar Sistema multi-agente
- Iniciar Consulta los posibles vectores de ataques
- Iniciar Casos de la base de conocimiento inicial
- Tomar o rechazar casos para formar parte de la base de conocimiento

- Finalizar Casos para su posterior procesamiento
- Finalizar Consulta de vectores de ataque

En la figura 7, se encuentran las funcionalidades asociadas al agente Poller.

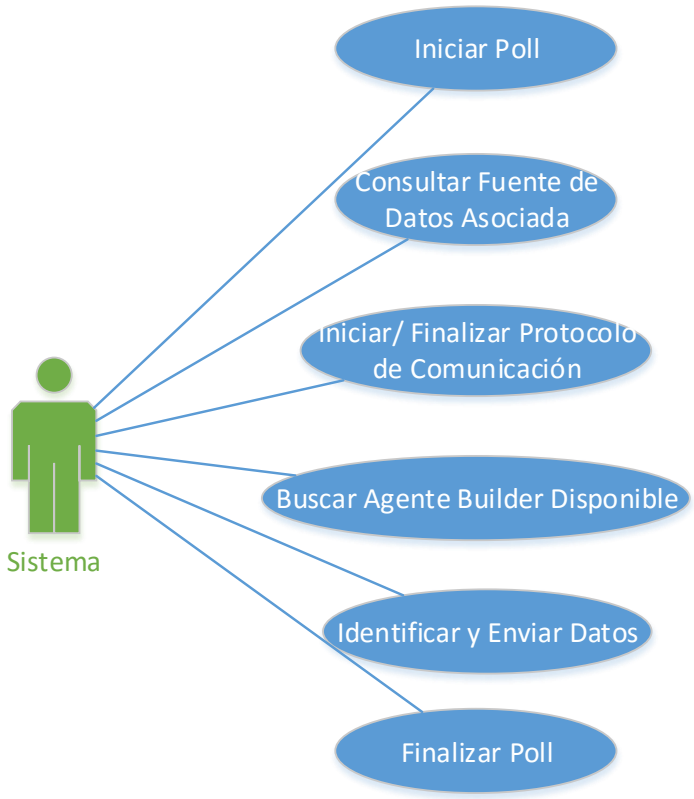

Figura 7. Modelo Casos de Uso Agente Poller

Para este modelo se presenta la interacción del Agente *Poller* encargado de obtener los diferentes casos que formaran los vectores de ataque y casos de base. Adicional, este agente se soporta en su interacción con los agentes de construcción para determinar la mejor opción en el momento de realizar el procesamiento asociado.

En la figura 8, se presentan las funcionalidades asociadas al Agente Builder.

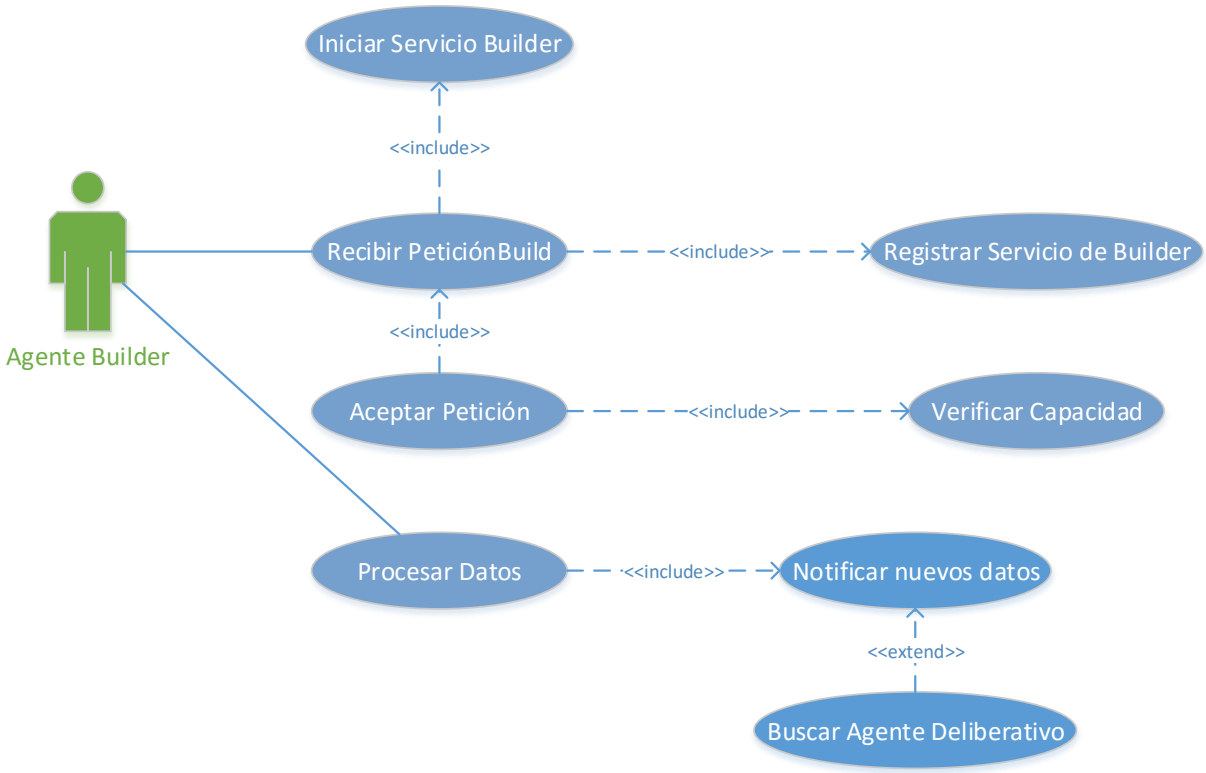

Figura 8. Modelo de Caso de Uso Agente Builder

El Agente *Builder* o de construcción, tiene uno de los roles más relevantes dentro del sistema, debido a que, ofrece de manera transparente y mediante su registro en las páginas amarillas, servicios de construcción de casos, a partir de los datos obtenidos por el Agente *Poller*. Una vez ha procesado los casos es encargado de notificarle al agente deliberativo para que este pueda ejecutar la búsqueda de nuevos casos con base a su previa configuración y base de conocimiento. Adicional determina su capacidad de procesamiento dependiendo de su capacidad de procesamiento actual.

En la figura 9, se presentan las funcionalidades asociadas al Agente deliberativo.

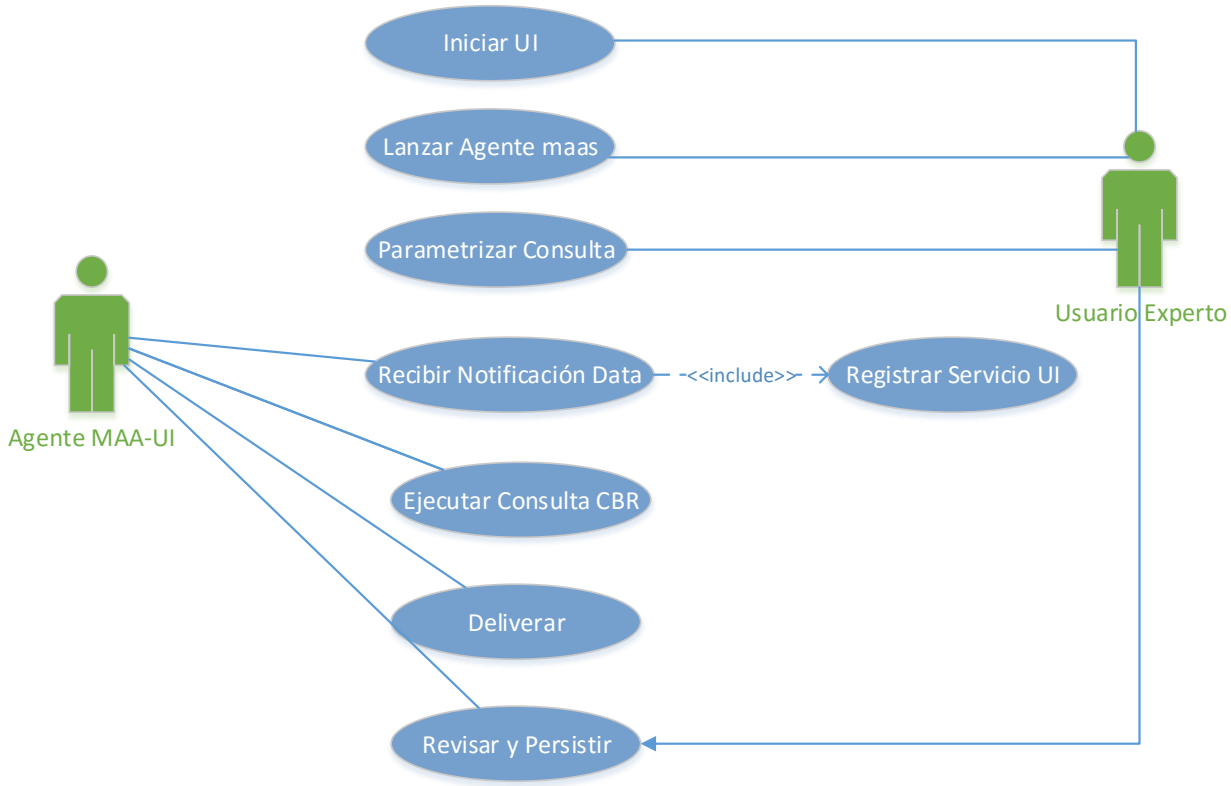

Figura 9. Modelo de Caso de Uso Maa-UI

En la anterior figura 9, se presentan las funcionalidades más relevantes a la implementación del Agente encargado del proceso de deliberación y de manejar la interfaz gráfica, su funcionamiento se basa en:

1. Controlar de manera global la interfaz gráfica, la visualización de casos, los diferentes comportamientos del agente y la recepción de notificaciones por parte del Agente *Builder*.
2. La generación de casos base por el agente constructor y en la utilización del algoritmo KNN para la clasificación y obtención de los casos cuya similitud sea mayor, acorde a los criterios configurados.

3. La deliberación y persistencia de los nuevos casos que cumplan con el criterio de clasificación
4. Configuración del criterio de clasificación para la utilización del algoritmo **KNN** para la obtención de los 5 primeros casos.

En la figura 10, se muestra de manera general y jerárquica las responsabilidades de cada agente. En puntos posteriores se explicará al detalle el rol de cada uno.

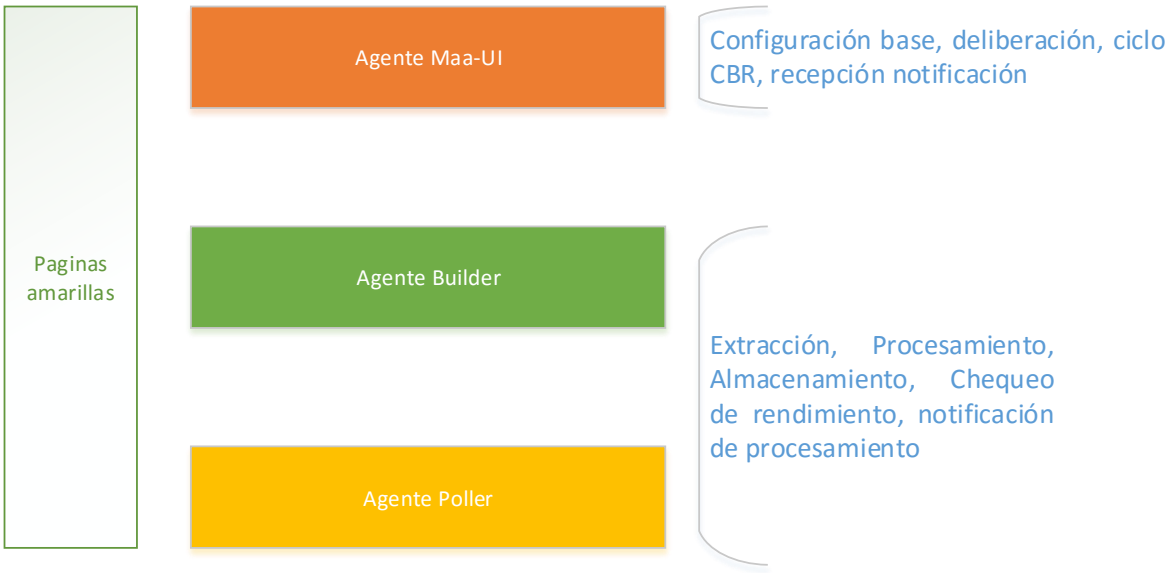

Figura 10. Diagrama General de Componentes

3.2.2.1.1 Definición de Roles y Agentes.

Las principales actividades que se dan dentro del sistema multi-agente pueden visualizarse en la Figura 11.

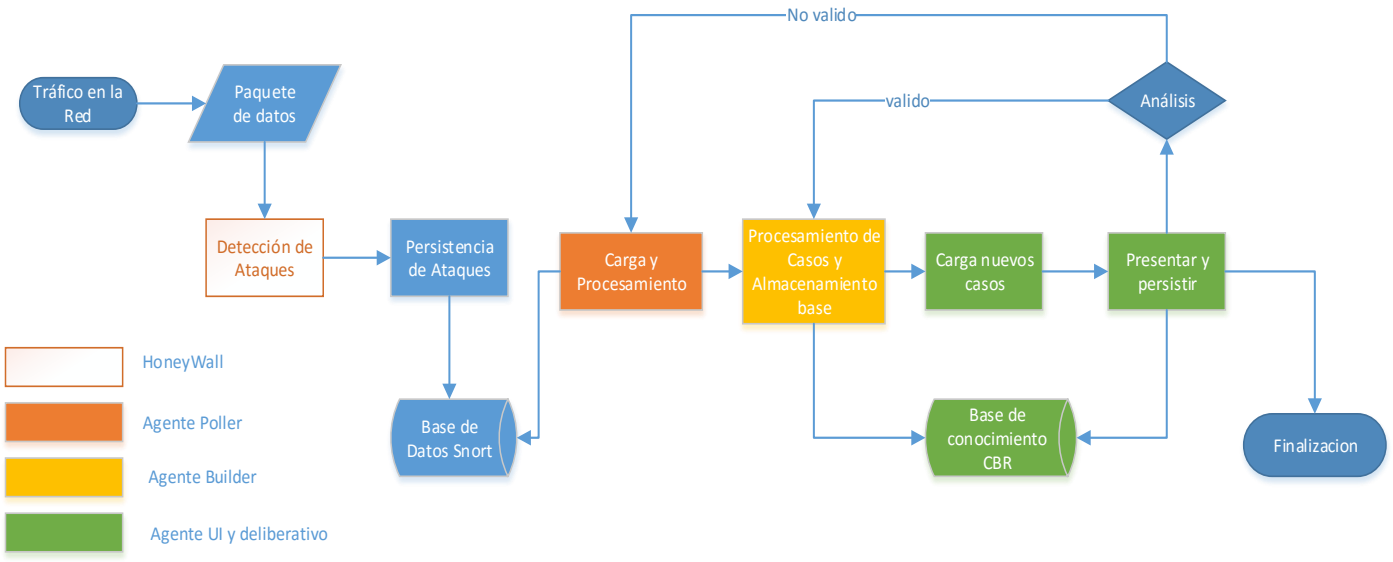

Figura 11. Diagrama Flujo Básico CBR+MAS (Tareas Generales del Sistema)

En la tabla 4, se presentaran las responsabilidades, propósitos, percepciones, tareas y salidas de los agentes que intervienen en el sistema.

Tabla 4 *Responsabilidades Agente Poller.*

Agente Sensor (*Poller*)	
Responsabilidades	Se encarga de sensar y obtener paquetes capturados previamente por el ambiente dentro de la *HoneyWall*. Este agente es clonado y distribuido dentro de las diferentes *HoneyWalls*, registrándose en cada caso en las *YellowPages*.
Propósito	Obtener datos de la fuente de información definida para la *HoneyWall*, en este caso *Snort*. Enviar la data enriquecida a un procesamiento posterior y determinar que agente puede recibir la petición.
Percepciones	1. Eventos de ataque capturados por *Snort* 2. Mensajes de otros agentes
Tareas	1. Consultar los diferentes posibles ataques identificados 2. Enriquecer y dar formato a la data consultada

	3. Identificar un agente que pueda recibir la data para su procesamiento 4. Determinar y enviar la data dependiendo de la respuesta de los agentes
Salidas	1. Data en formato **JSON** para ser enviada 2. Data enriquecida

Tabla 5 *Responsabilidades Agente Builder.*

Agente Constructor (*Builder*)	
Responsabilidades	Se encarga de sensar y recibir la data referente a posibles ataques o comportamientos anormales identificados previamente por agentes *Pollers*, adicional notifica al agente deliberador sobre nuevos datos. Este agente es clonado y distribuido de manera estratégica de acuerdo a la carga y posición geográfica asociada
Propósito	Procesar la data que es enviada por otros agentes basados en la carga actual para persistir a data acorde a patrones definidos como casos a ser utilizados dentro de la fase de comparación del razonamiento basado en casos.
Percepciones	1. Data procesada de posibles ataques 2. Mensajes de otros agentes
Tareas	1. Recibir data para su conversión y persistencia 2. Definir su capacidad de procesamiento para ofrecer su servicio 3. Procesar la data acorde al lineamiento definido para su reutilización dentro del proceso de identificación de nuevos vectores de ataque 4. Registrarse en las *YellowPages*. 5. Notificar al agente deliberativo sobre nuevo datos
Salidas	1. Data almacenada en base de datos 2. Data procesada como casos con solución y descripción

Tabla 6 *Responsabilidades Agente Maa-UI deliberativo.*

Agente UI Deliberativo	
Responsabilidades	Se encarga de sensar y recibir notificaciones sobre nuevos ataques procesados por el agente *Builder*. Adicional se encarga

	de controlar la interacción entre el **UI** y la lógica propia del Agente, deliberar y persistir nuevos casos.
Propósito	Permitir la configuración base para la detección de nuevos vectores de ataques dentro, controlar la ejecución del agente, la deliberación sobre los ataques y la persistencia de los mismos.
Percepciones	1. Notificaciones de nuevos ataques por parte del agente *builder*.
Tareas	1. Recepción de notificación sobre nuevos ataques 2. Permitir la configuración de nuevos patrones de ataque para realizar la búsqueda en **CBR**. 3. Controlar la interacción entre la lógica de agente y la interfaz gráfica. 4. Realizar el proceso de deliberación con base a la información obtenida previamente y almacenada por el agente *Builder*.
Salidas	1. Nuevos patrones de ataque 2. Nuevos casos en la base de conocimiento

Dentro del contexto de la metodología **UPSAM** el diagrama **aUML**[30] correspondiente a los Agentes *Poller*, *Builder* y Maa-UI se presentan en las siguientes figuras: Ver figuras 12, 13 y 14.

Figura 12. Diagrama de Clases Agente *Poller*

[30] Lenguaje de Modelado, basado en UML tradicional, pero focalizado a agentes y su interacción.

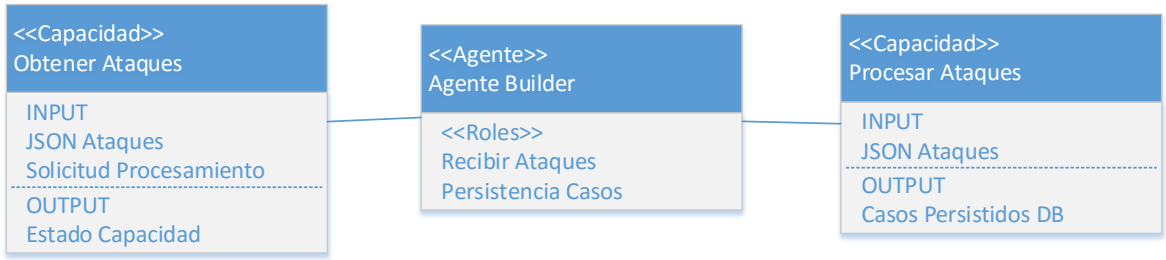

Figura 13. Diagrama de Clases Agente *Builder*

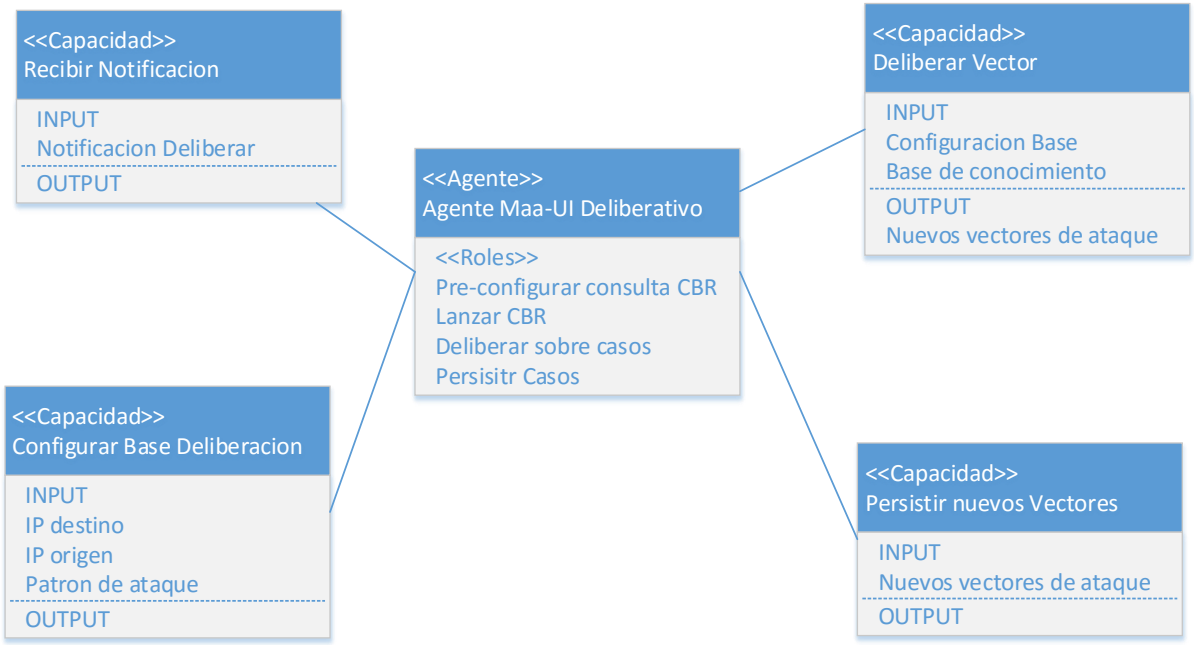

Figura 14. Diagrama de Agente Maas –UI Deliberativo

3.2.2.1.2 Modelo de Arquitectura y Organización de Agentes.

El modelo de Arquitectura que se plantea se sustenta en un planteamiento de sociedad asociado a los Agentes y la interacción que estos forman, con miras a conseguir objetivos globales. En la Figura 15, se especifican los diferentes componentes que interfieren en el sistema para conseguir un mismo fin, el cual se sustenta en la alimentación de una base de conocimiento para la búsqueda de nuevos patrones de ataque.

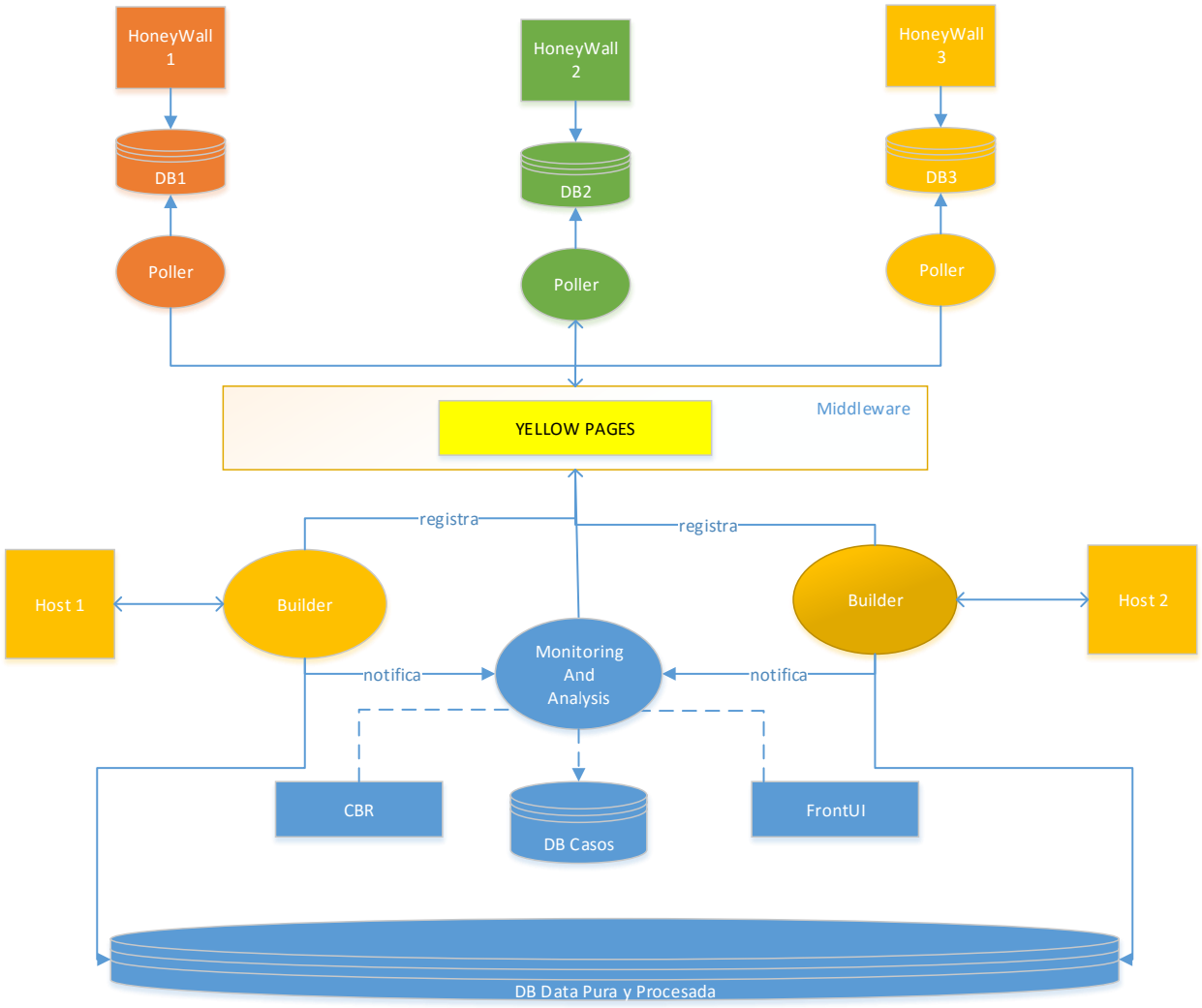

Figura 15. Diagrama Modelo de Arquitectura

En la Figura 16, se muestra el diagrama de secuencia general contemplado dentro de los contenedores específicos, identificando y especificando tareas que realizan los agentes para lograr su objetivo global.

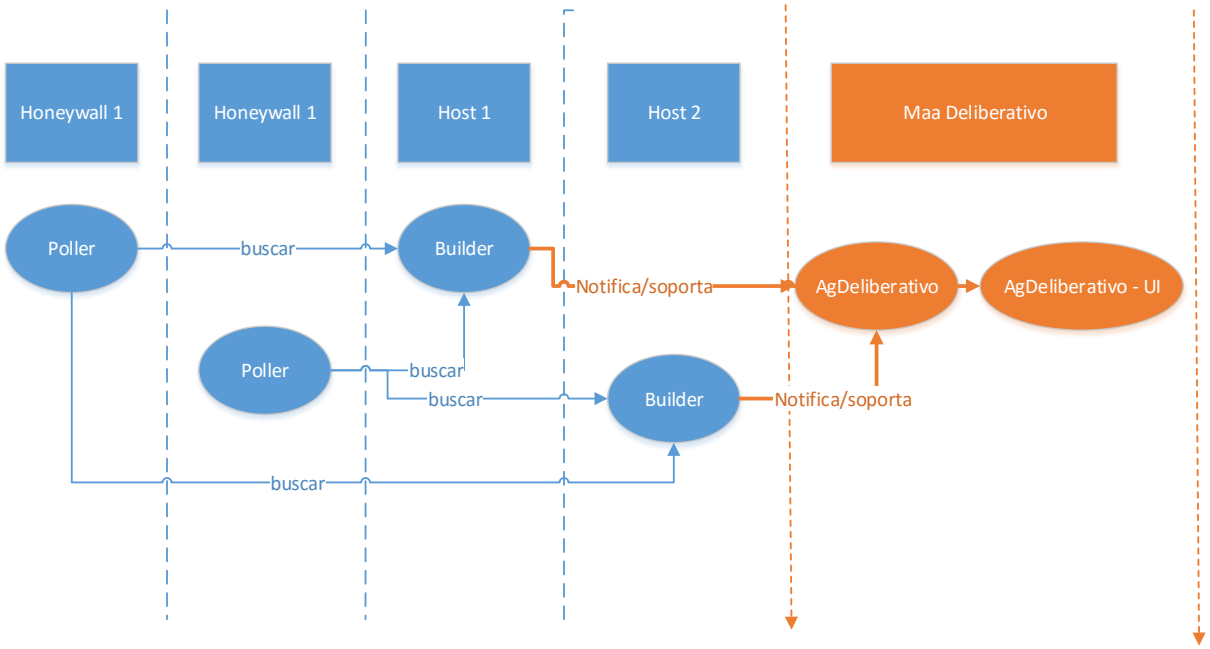

Figura 16. Diagrama Secuencia General

3.2.2.2.3 Modelo de Comunicación de los Agentes.

El modelo de comunicación trata de mostrar la secuencia de interacción entre los diferentes agentes, sin ser determinísticos debido a su capacidad de autonomía. A continuación en la figura 17, se muestra un acercamiento a la secuencia de mensajes que componen la comunicación entre los agentes *poller*, *builder* y demás componentes de la plataforma.

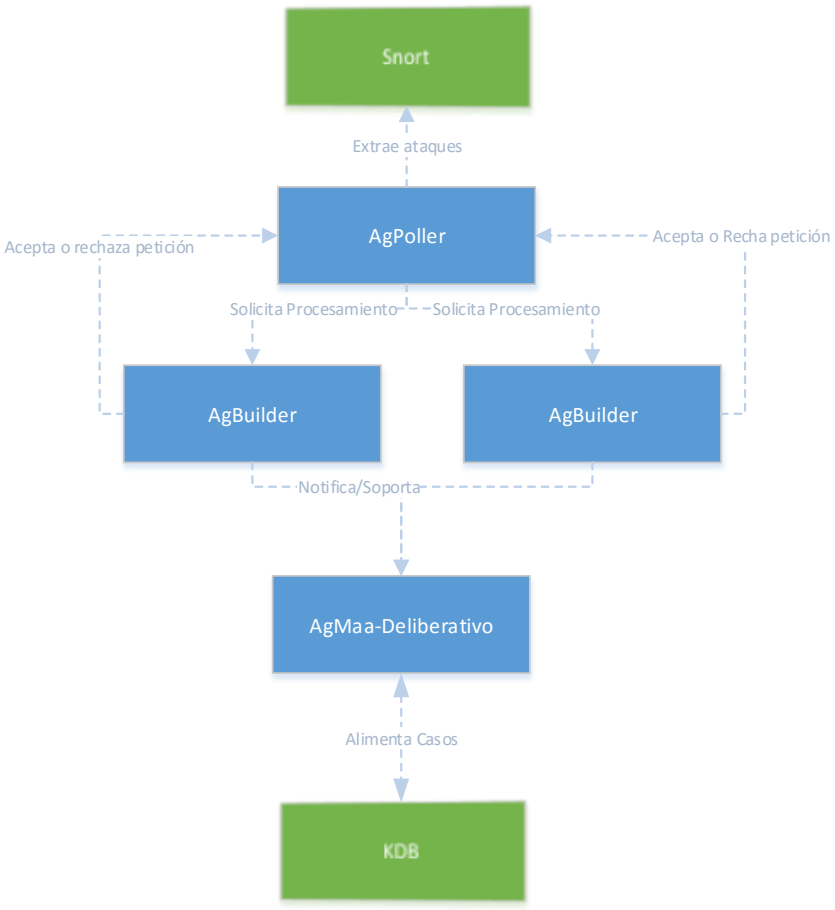

Figura 17. Diagrama Modelo de Comunicación

En la figura 18, se puede apreciar el diagrama de secuencia entre los diferentes componentes.

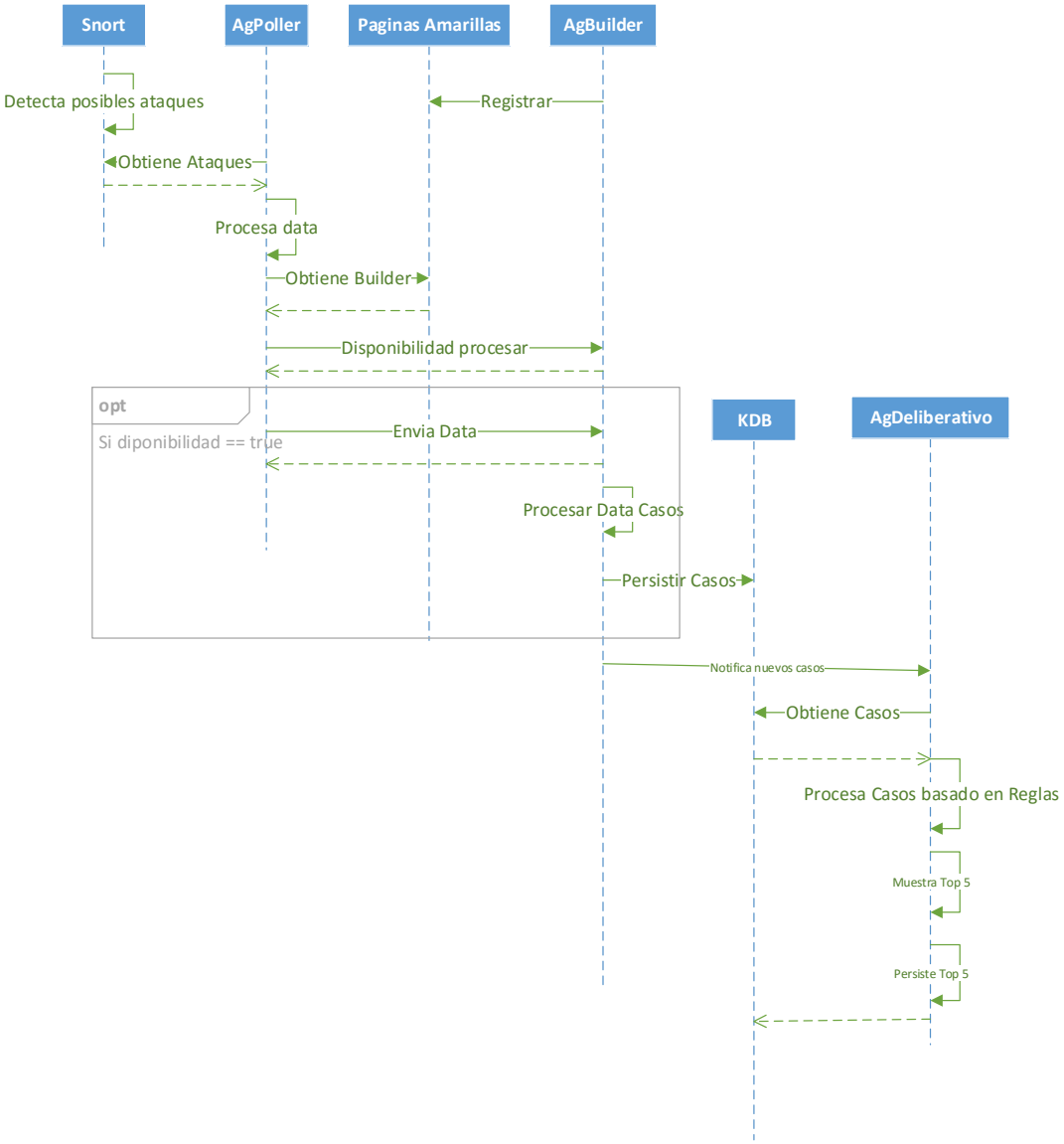

Figura 18. Diagrama de Secuencia - Agentes

3.2.2.1.4 Modelo interno de Agentes.

El modelo interno de agentes representa de manera jerárquica la relación entre las diferentes clases que permiten definir las características a un agente específico. En la Figura 19, se presenta un modelo base en donde se muestran las diferentes clases de apoyo de los agentes.

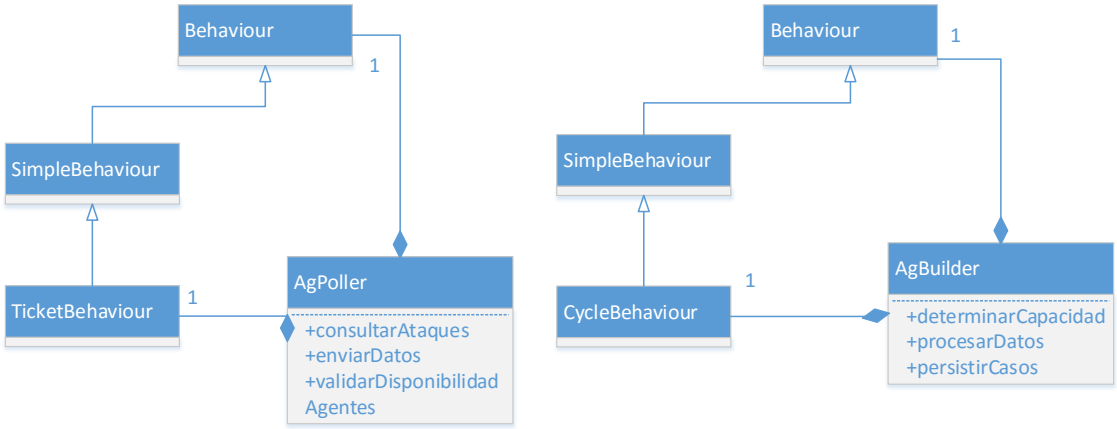

Figura 19. Diagrama de Clases de clases de Soporte

En la figura 20., se muestra el modelo interno de agente.

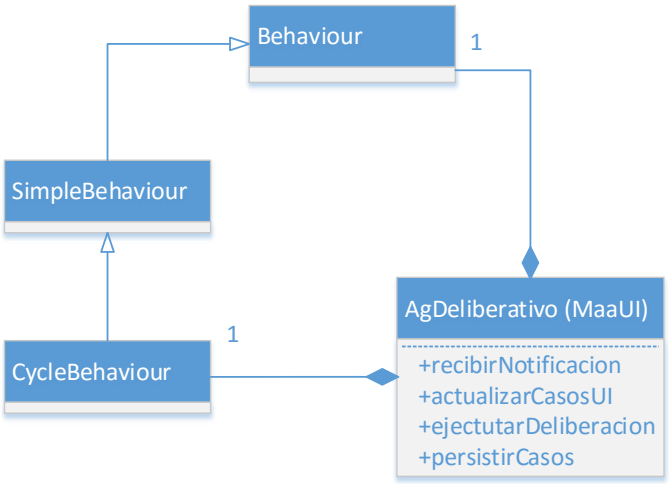

Figura 20. Modelo Interno de Agentes / Modelo de Comportamiento

3.2.2.1.5 Modelo de Recursos.

El siguiente modelo representado en la figura 21, se muestra el soporte a los diferentes agentes a nivel de recursos tecnológicos, aplicaciones, librerías y demás componentes base.

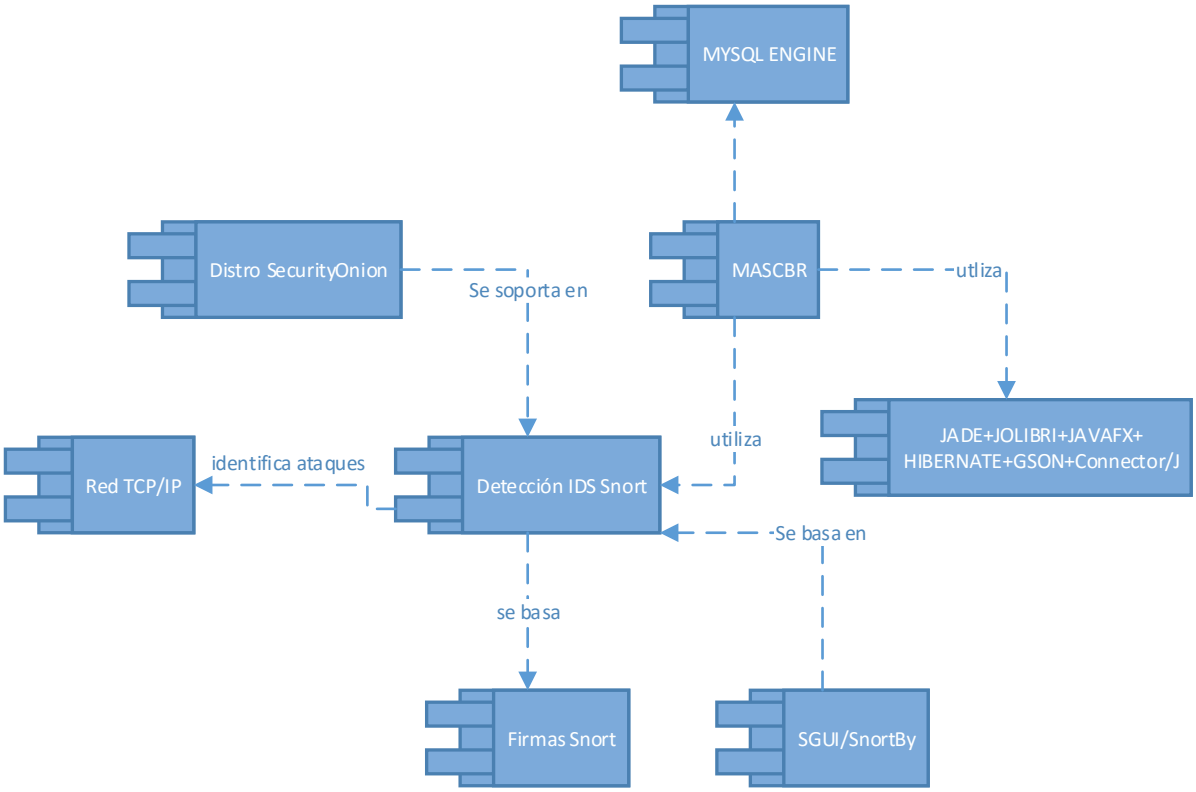

Figura 21. Modelo de Recursos

3.2.2.1.6 Explicación Entorno Controlado.

En la figura 22, se puede observar la topología asociada al laboratorio que se construyó como se muestra a continuación.

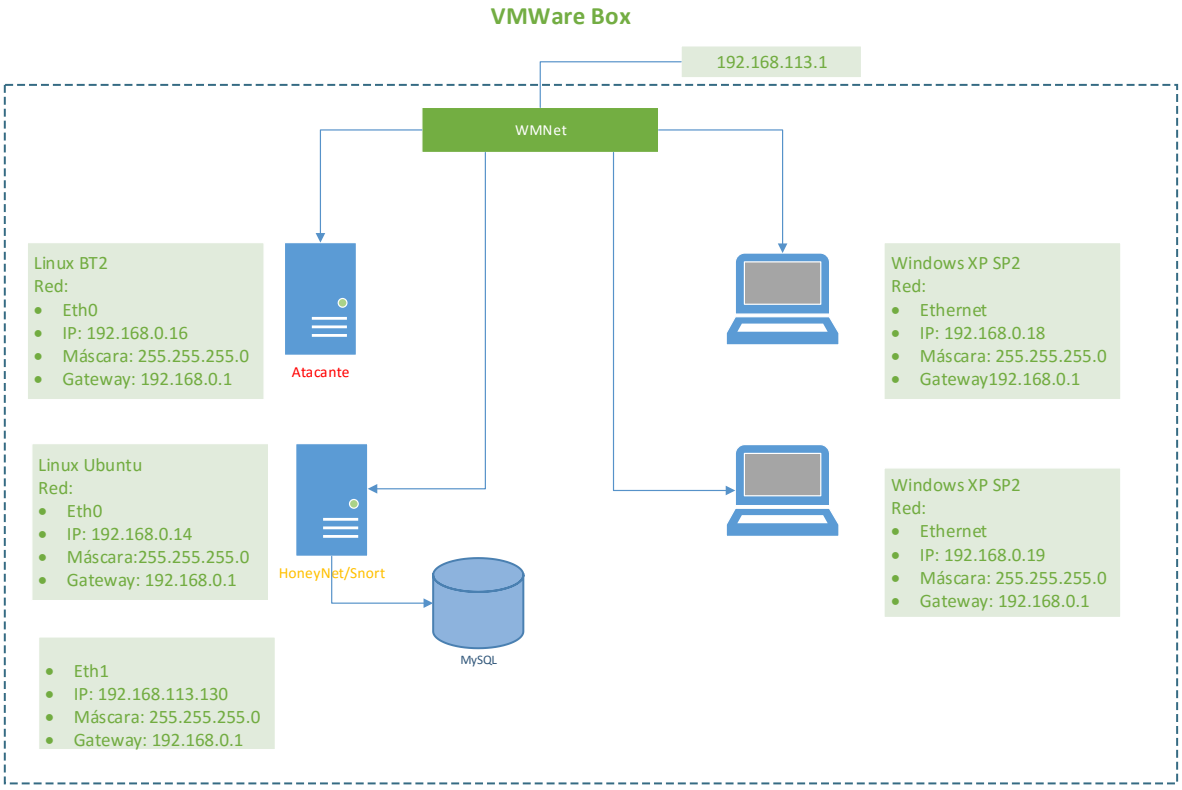

Figura 22. Topología de Red Planeada

Toda la solución para generar un entorno controlado y vulnerable utilizó la plataforma *VMWare Workstation*, cuyo objetivo fue servir de medio para virtualizar las máquinas necesarias para definir el entorno. La solución consta de los siguientes elementos:

- **Máquina Atacante**: Esta máquina fue virtualizada con el único objetivo de servir como elemento base, desde el cual se realizarían ataques a máquinas vulnerables dentro del laboratorio. La ficha técnica de esta máquina se define:

Tabla 7 *Características de la Máquina Atacante.*

Característica	Descripción
Sistema Operativo	Backtrack 5r3
Memoria	512MB

Procesadores Virtuales	1
Almacenamiento	20GB
Adaptador de Red	Brideged Eth0: 192.168.0.15 Máscara: 255.255.255.0 Gateway: 192.168.0.1

- **HoneyWall:** Esta máquina fue virtualizada para que actuará como un punto de captura centralizada dentro de la red de posibles ataques, su objetivo es permitir el control de los datos de entrada que pueden indicar un ataque, capturar estos datos, analizarlos y permitir herramientas para analizarlos. La ficha técnica de esta máquina se define:

Tabla 8 *Característica de la máquina HoneyWall.*

Característica	Descripción
Sistema Operativo	SecurityOnion 12.04
Memoria	1GB
Procesadores Virtuales	1
Almacenamiento	20GB
Adaptador de Red	Brideged Eth0: 192.168.0.17 Máscara: 255.255.255.0 Gateway: 192.168.0.1
Adaptador de Red 2	Custom (VMNet1) Eth1:192.168.113.131 Máscara:255.255.255.0 Gateway:192.168.0.1

- **Máquina(s) Vulnerable(s):** Esta máquina fue virtualizada para representar una o varias máquinas vulnerables, desde su sistema operativo hasta las aplicaciones que corren en esta. La ficha técnica asociada se presenta:

Tabla 9 *Características de Máquina Vulnerable.*

Característica	Descripción
Sistema Operativo	Windows XP SP2
Memoria	512MB
Procesadores Virtuales	1
Almacenamiento	40GB
Adaptador de Red	Brideged
	Eth0: 192.168.0.18 Máscara: 255.255.255.0 Gateway: 192.168.0.1

Esta máquina es fácilmente clonada dentro de la infraestructura definida en caso de ser necesario.

3.2.2.1.7 Estructura de Datos de Snort y SecurityOnion.

Una vez un posible comportamiento y/o ataque malicioso es identificado, *snort* persiste este en su base de datos como un evento, garantizando que la información asociada sea lo suficientemente clara para realizar una clasificación y seguimiento, sin embargo, su integridad referencial hacia otras tablas que son complementarias no existe de manera explícita limitando el modelamiento automático de este.

Para este proyecto, existen tres tablas específicas que soportan e identifican la captura de los diferentes eventos los cuales se pueden observar en la figura 23:

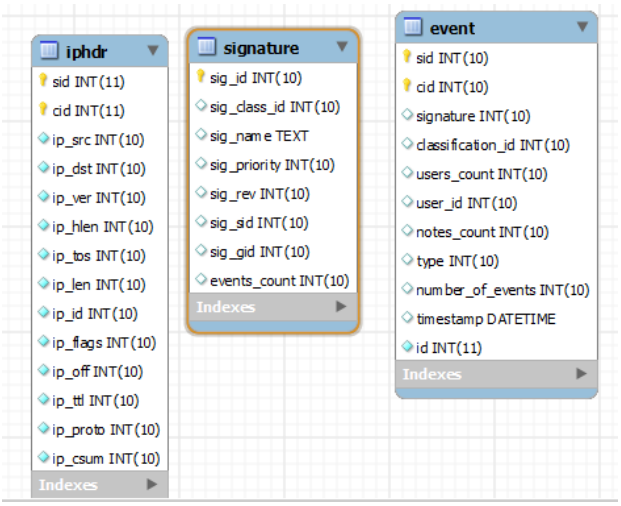

Figura 23. Tablas Base de Eventos

Event: Almacena los eventos que son capturados por el IDS.

Signature: Almacena la información base sobre las firmas asociadas a los posibles eventos.

Iphdr: Almacena la información sobre el protocolo IP de los eventos asociados.

En la figura 24, se observa el diagrama completo de las tablas asociadas es el siguiente:

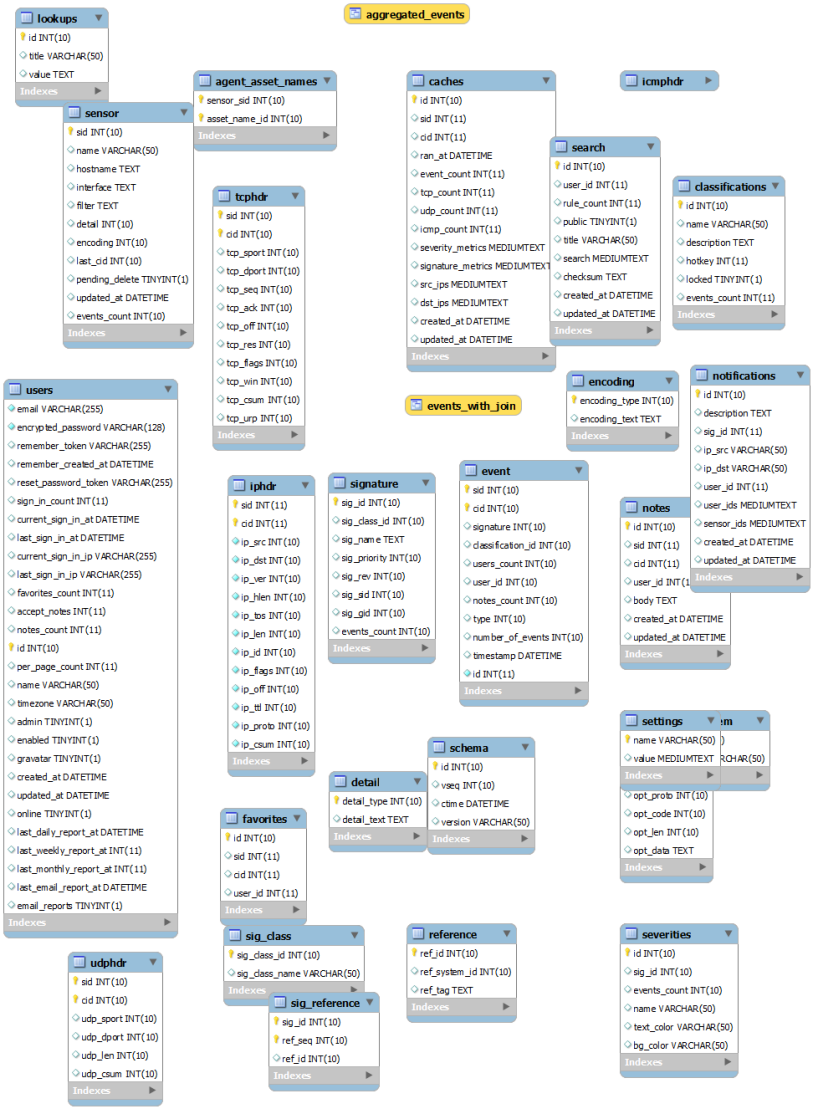

Figura 24. Diagrama E/R *Snort/Security Onion*

3.2.2.1.8 Explicación Backtrack.

Dentro del laboratorio controlado se utiliza esta distribución de Linux basada en Ubuntu, cuyo objetivo es proveer las herramientas necesarias para realizar un ataque a una máquina específica.

Para este proyecto, se utilizan las siguientes herramientas contenidas en este:

- **Metaexploit:** Proyecto *OpenSource* que permite el desarrollo y la ejecución de *Exploits*[31] en una máquina objetivo, en donde previamente se ha identificado una vulnerabilidad. En la figura 25 (Highsec, 2013) define su arquitectura de la siguiente forma:

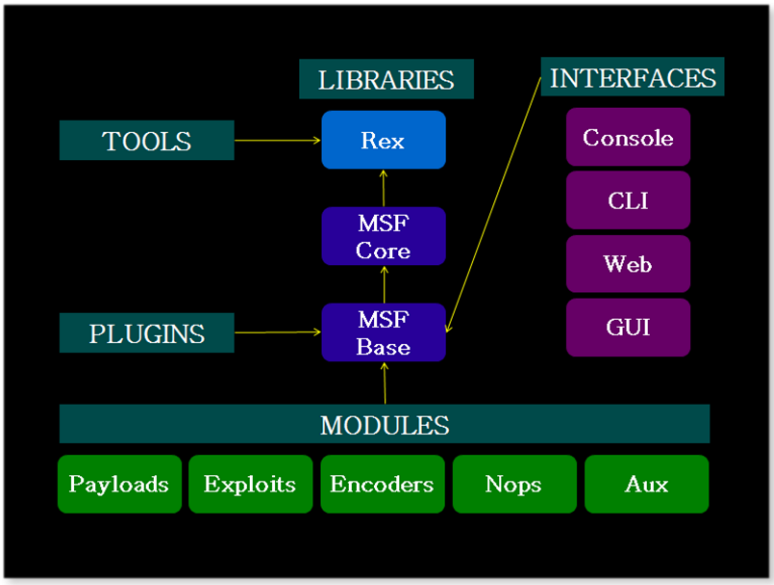

Figura 25. Arquitectura Base Metaexploit. (Highsec, 2013)

- **Nmap:** Proyecto *OpenSource* que permite el escaneo de máquinas dentro de una red, en donde el objetivo es el descubrimiento de máquinas, sistema operativo, puertos
- **Wireshark:** Proyecto *OpenSource* utilizado como herramienta de diagnóstico, análisis y captura de paquetes de Red.

Los agentes que se desarrollaron durante el proyecto fueron implementados utilizando la plataforma JADE, que ofrece el marco base para su desarrollo.

[31] Componente de software que se encarga de aprovechar una vulnerabilidad en una plataforma especifica

3.2.2.2 Desarrollo.

El objetivo que se intenta lograr durante la especificación del desarrollo es mostrar y detallar los elementos base y complementarios que soportan la plataforma propuesta.

3.2.2.2.1 Herramientas de soporte a la implementación.

Para llevar a cabo la implementación asociada a la plataforma propuesta se utilizaron las siguientes herramientas:

- **Java:** Lenguaje compilado orientado a Objetos que corre sobre una máquina virtual que se encarga de interpretar el código a una arquitectura especifica.
- **JavaFX:** Plataforma de software creada para entregar aplicaciones ricas para internet, cuya intención fue reemplazar a la antigua tecnología *Swing* de *Java*.
- **Hibernate:** Herramienta de mapeo objeto relacional.
- **MySql:** Motor de base de datos que soporta tanto la base de conocimiento como al IDS.
- **Nmap:** Herramienta gratuita utiliza dentro del proceso de descubrimiento de información
- **Gson:** Librería gratuita liberada por *Google* para la manipulación de objetos JSON.
- **Snort:** IDS que soporta la base de conocimiento de la plataforma
- **Metaexploit:** Proyecto de código abierto que permite la explotación de vulnerabilidad en sistemas y plataformas.

3.2.2.2.2 Comportamiento y entorno de la plataforma IDS.

- Generación de alertas e identificación de eventos: Los paquetes capturados por *Snort* son comparados contra una firma existente y definida dentro de la base de este. En la

figura 26 (Isaza, 2010) define en su documento doctoral la representación ontología de la estructura de una regla.

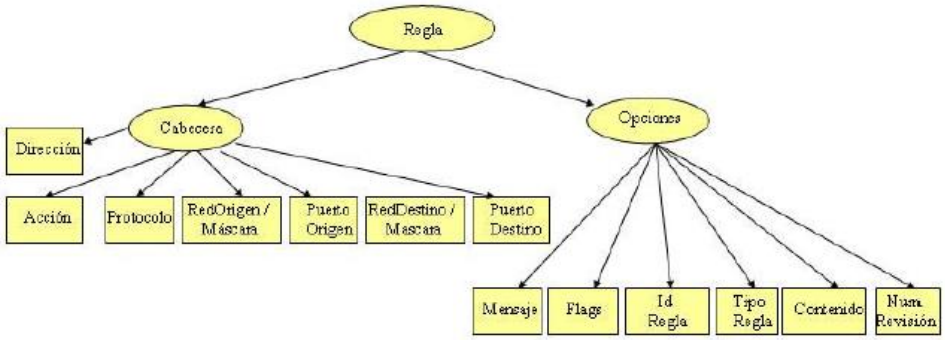

Figura 26. Ontología de una firma de Snort. (Isaza, 2010)

Para la implantación de la plataforma se utilizan las reglas publicadas a la comunidad por el grupo VRT (*Vulnerability Research Team)*[32]. El siguiente ejemplo muestra una de las firmas definidas dentro de la simulación de ataques en el entorno controlado.

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"PROTOCOL-ICMP PING undefined code"; icode:>0; itype:8; metadata:ruleset community; classtype:misc-activity; sid:365; rev:11;)

La estructura anterior se divide en la definición del encabezado y las opciones de la regla.

La estructura del encabezado define las características a nivel de red que identifican el comportamiento inicial de la alerta; la estructura de las opciones asociadas a la alerta determina

[32] Grupo conformado por un equipo de expertos en seguridad informática cuyo objetivo es reaccionar de manera proactiva y reactiva a los últimos ataques. Adicional, son los encargados de publicar de manera recurrente el listado de firmas de Snort.

mensajes e información para determinan en que segmento del paquete se debe realizar la inspección correspondiente y definir si se realiza alguna acción o no.

En la figura 27, se representa el administrador grafico SGUIL, para la gestión de eventos. Dentro de su configuración exige que se le indique bajo que interfaz va a escuchar el tráfico, en este caso, se decide debido a la configuración del entorno que sea la interfaz eth0 de la máquina *HoneyWall*:

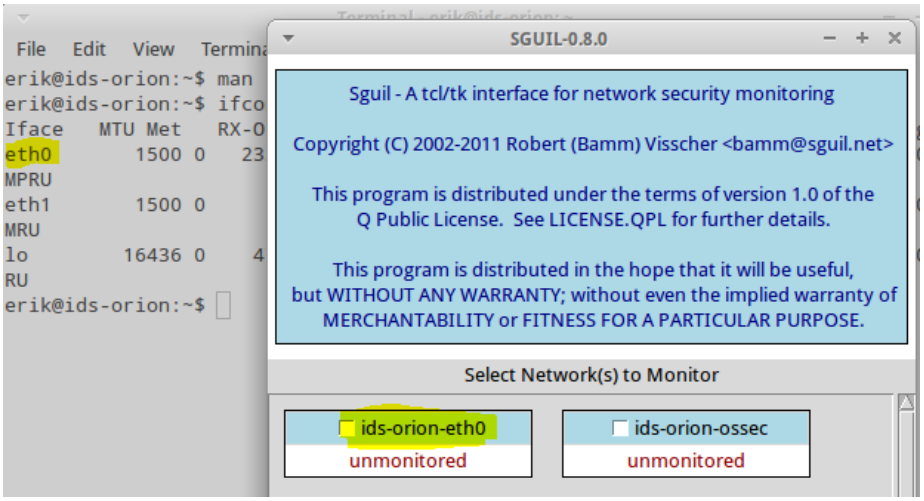

Figura 27. Configuración Inicial SGUIL

En total se simularon y capturar 2161 eventos en el entorno controlado, dentro de estos eventos se pueden mencionar el descubrimiento de información a través del uso de *nmap*, la explotación de vulnerabilidades con *metaexploit*, identificación de firmas y aplicativos de *back/front*.

Adicional a lo anterior y durante la ejecución se realizaron capturas reales de ataques a maquinas internas de la red, eventos que fueron capturados por el IDS. Para la visualización de todos los eventos se utilizó *snorby*.

En la figura 28, se muestra la interfaz del Dashboard Snorby.

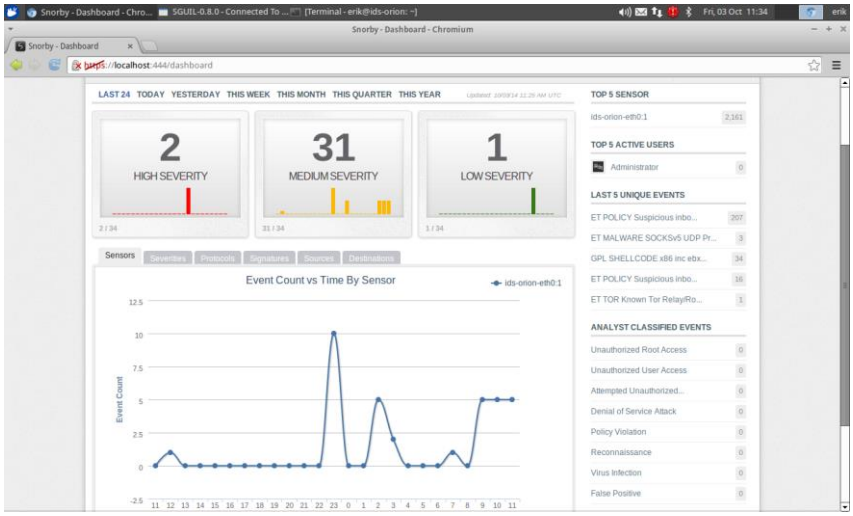

Figura 28. Dashboard Snorby

Como se puede observar en la figura 28, las dos alertas de seguridad máxima fueron ejecutadas desde una red externa al entorno controlado y dirigidas a una de las máquinas internas de la red.

La plataforma utilizada para el IDS presenta inicialmente un arquitectura de un nivel, en donde tanto el *front*, componentes de capa media e integración, componentes de persistencia se encuentran en un mismo nodo, adicional, se soporta un motor de base de datos *MySql* en modo *Stand Alone,* sin tener ningún tipo de estrategia de *cluster*.

En este punto, es importante tener en cuenta que todos los componentes asociados a *Snort* a nivel interno y cada una de sus capas (Decodificador, Máquina de detección, servicio de auditoria y alertas) se encuentran desplegados en el mismo nodo.

Snort, utiliza internamente en su máquina de detección el algoritmo de Booyer-More[33] para búsqueda de patrones en un texto determinado, esto lo hace mediante una búsqueda de derecha a izquierda bajo la siguiente premisa:

$$Si\ P = TEST\ y\ T = CASJ, n\ y\ m\ definen\ la\ longitud\ de\ ambos$$

$$Siendo\ Y(m) = J\ y\ X(n) = T, Y(m) \neq X(n)$$

Con base en lo anterior, se evidencia que ya no es necesario realizar una comparación de $Y(m-1)\ con\ Y(n)$ ya que nunca se va a encontrar el patrón en el texto. Adicional, el algoritmo hace uso de estrategias de intercambio de caracteres erróneos para evitar la repetición de comparaciones contra un carácter específico y el intercambio de caracteres correctos para la alineación de los caracteres del patrón con los ya encontrados.

Es debido a lo anterior que *Snort* es una excelente opción dentro de la implementación de una *HoneyNet*, alineándose al proyecto y ofreciendo grandes ventajas, por esto se escogió *Snort* como IDS.

3.2.2.2.3 Arquitectura, interacción y funcionamiento de la plataforma MAS+CBR.

1. Arquitectura de la plataforma: Para la implementación de la plataforma MAS+CBR se definió una arquitectura en 4 niveles con opción de escalabilidad horizontal de manera transparente como se puede observar en la figura 29.

[33] Algoritmo búsqueda de cadenas, el cual basa su funcionalidad en el sondeo de coincidencias, marchando hacia atrás

Figura 29. Arquitectura de la Plataforma

Aun cuando se define de la manera antes mencionada, el despliegue físico de las máquinas en diferentes niveles hace que se garantice una escalabilidad horizontal inherente a la arquitectura propuesta. De manera genera se puede especificar en cada nivel el componente asociado:

- Sistema de razonamiento y Gestor de conocimiento
    - Interfaz de usuario que permite la gestión y configuración del CBR
    - Elementos de negocio que garantizan la interacción hacia los demás componentes
    - Elementos de integración con otras plataformas y/ componentes
    - Elementos de persistencia que trabajan hacia la base de datos
- Agente *Builder* o *Constructor* y procesador de datos

    - Elementos propios de la definición del agente
    - Elementos de negocio que garantizan la interacción con otros componentes
    - Elementos y servicios comunes a los agentes
- Agente *Poller* o *Extractor* de datos.
    - Elementos propios de la definición del agente
    - Elementos de negocio que garantizan la interacción con otros componentes
    - Elementos y servicios comunes a los agentes

Los demás componentes que se definen en los diferentes niveles poseen una arquitectura propia en n-capas y el diseño propio de cada componente de la plataforma fue especificado en el diseño de la misma.

En la figura 30, se puede observar como el diagrama muestra de manera general la integración entre los diferentes componentes de la plataforma.

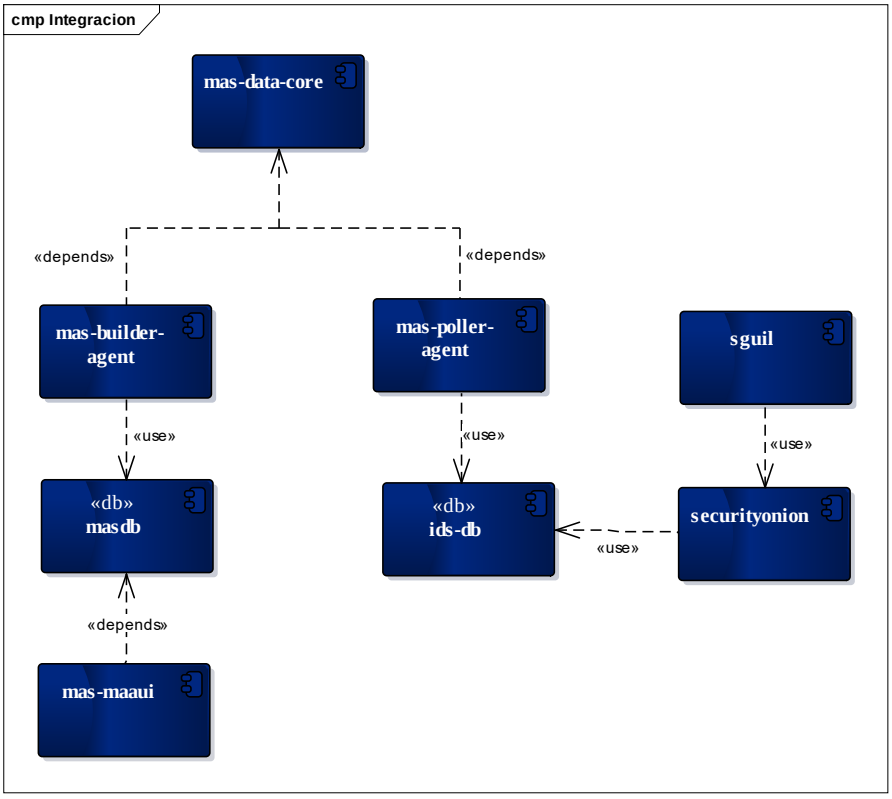

Figura 30. Diagrama de Componentes Plataforma

2. Comportamiento y estructura del componente de capa media JADE: Este se utiliza como núcleo y gestor de la plataforma MAS (Multi Agent System), sus características de capa media y de gestión del protocolo FIPA para permitir la interacción de los agentes bajo un contexto social hace que cumpla las necesidades básicas para la plataforma. Jade propone una arquitectura en n-niveles, siendo cada nivel un posible host de un agente específico que incluye un entorno de ejecución completo, como lo presentan (Bellifemine, Caire, & Greenwood, 2007) a continuación en la figura 31.

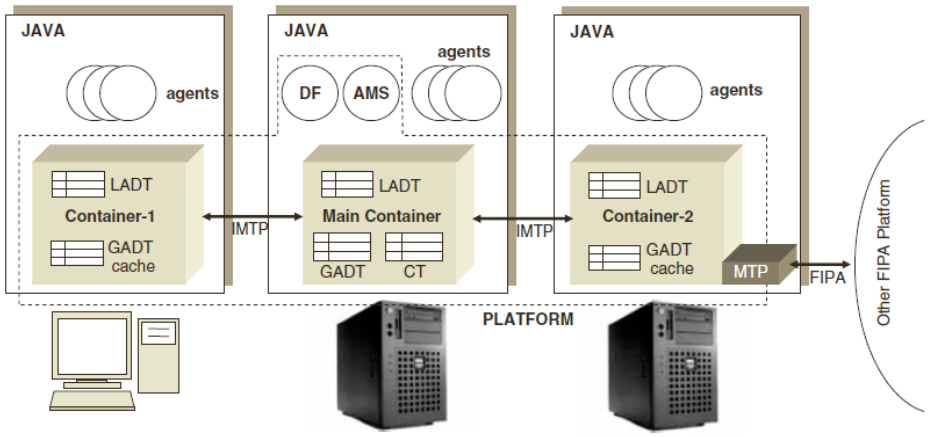

Figura 31. Arquitectura Jade (Bellifemine, Caire, & Greenwood, 2007).

Dentro de esta definición siempre se va a tener un contenedor principal, el cual se responsabiliza por de los otros contenedores y es en el cual se realiza la búsqueda de los diferentes agentes, mediante el uso de su DF[34] (Directory Facilitator).

Para nuestro caso, JADE está corriendo en nuestro Nivel 3 en donde se encuentra como componente de capa media ofreciendo los servicios de AMS[35] (Agent Management System) para la gestión de los agentes *Poller* y *Builder*, el servicio de DF encargado proveer las funcionalidades de páginas amarillas en donde todos los agentes ofrecen y consumen los servicios propios de su dominio. En este componente, se encontrarán registrados los agentes que proveen los servicios de extracción de datos y procesamiento, garantizando transparencia en la locación, crecimiento transparente para los agentes usuarios de los servicios expuestos, fácil despliegue en diferentes locaciones geográficas debido a un punto centralizado de registro.

[34] Componente entre la arquitectura de JADE, encargado del registro y publicación de agentes basados en sus servicios.
[35] Componente entre la arquitectura JADE, encargado de la gestión de los agentes.

JADE dentro de su núcleo, implementa la especificación FIPA y su protocolo de comunicación que sirven como base de interacción entre los agentes propuestos, su fundamento se basa en el siguiente planteamiento:

En la figura 32, podemos ver como (Poslad, 2007) presenta el estándar de un mensaje FIPA.

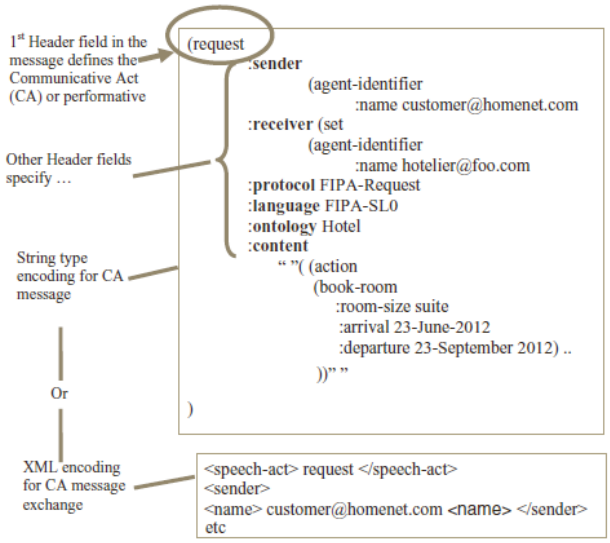

Figura 32. Estructura mensaje FIPA – ACL (Poslad, S. (2007)

El estándar que se visualiza en la figura 32 es el que JADE, en los mensajes asociados a la interacción entre los diferentes agentes. Esto se podrá visualizar durante la ejecución de la plataforma y el uso del Agente *Sniffer*.

JADE en su ejecución inicial corre tres componentes o agentes que le dan las características previamente mencionadas, estas son:

- AMS (*Agent Management System*): Es el sistema encargado del servicio de nombramiento, encargado de que cada agente tenga un único nombre. Este

también representa un controlador de agentes, permitiendo la terminación y creación de los mismos.

- DF (*Directory Services*): Es el sistema encargado de ofrecer un punto centralizado de publicación/consumo de servicios que son ofrecidos por los diferentes agentes. Oficialmente se conoce este punto como las páginas amarillas
- RMA (*Remote Monitoring Service*): Ofrece un entorno gráfico para el control de agentes y monitoreo.

Internamente JADE acorde a la especificación FIPA define los estados en los cuales un agente puede encontrarse y los diferentes comportamientos que se le pueden dar, entendiendo por esto la actividad necesaria para lograr un objetivo, la cual se ejecuta hasta que se termine. Estos estados y el control de estos agentes pueden ser visualizados desde la consola RMA.

Dentro de los comportamientos especificados para los agentes que se definen dentro de la plataforma JADE, se resaltan los siguientes que fueron utilizados durante la implementación:

- *TicketBehaviour:* Permite la programación recurrente de una actividad para el agente, este tipo de comportamiento define una ejecución sin una terminación definida.
- *Behaviour:* Este comportamiento es genérico y define dentro de su flujo un estado, el cual define el flujo y terminación del agente.

- *CycleBehaviour:* Este comportamiento nunca termina y garantiza la ejecución de su método *action().*

Los comportamientos que se mencionan definen a los agentes que se utilizan las capacidades necesarias para obtener el objetivo común, suplir los datos para la base de conocimiento que suplirá el CBR.

3. Comportamiento Agentes Plataforma: Los agentes definidos dentro del desarrollo de la plataforma basan su objetivo común en la recolección y procesamiento de los eventos capturados previamente en redes trampa señuelo, esto por medio de un componente de detección de intrusos, en este caso, *Snort*. Una vez cada agente realice su parte para obtener el objetivo común, los eventos procesados serán base de conocimiento para que la aplicación de razonamiento pueda deliberar, aportar y aprender de manera constante.

- **Agente *Poller*:** El primer agente desarrollado nace de la necesidad de un componente ligero, transversal a la arquitectura subyacente en la que se ejecute, que ofrezca movilidad y capacidades de interacción dentro de un entorno distribuido y que pueda crecer de manera transparente, eliminando la limitante de la locación geográfica del mismo. Una vez se cumplen las características mencionadas anteriormente, se contextualiza el dominio del agente a su interacción con *Snort*, especialmente con su esquema de datos para la extracción de los eventos dentro de un rango de tiempo, para esto, se adicionan comportamientos cuyo foco es garantizar la ejecución de actividades para obtener y procesar los datos obtenidos.

  Inicialmente, el comportamiento que se asigna al agente es *TicketBehaviour,* cuya programación recurrente garantiza en cada ejecución la búsqueda dentro del directorio de páginas amarillas ofrecido por el *Directory Facility* de JADE, agentes

cuyo servicio este tipificado como "*building-service*", esto realizado a través del DF, garantiza que indiferente de la locación geográfica se puedan ubicar los agentes asociados. Una vez se obtiene el listado de agentes bajo una tipología específica, se le adiciona un comportamiento genérico *Behaviour* cuya secuencia de pasos definidos por un estado, genera una secuencia de mensajes acorde al protocolo de comunicación FIPA-ACL bajo la siguiente secuencia:

- CFP (*Call for Proposal*): Se envía un mensaje cuyo objetivo es iniciar una conversación, en este caso, se solicita una propuesta para el servicio de construcción y procesamiento.
- *Propose*: Una vez que los agentes reciben y están interesados en ofrecer el servicio de construcción, cada uno realiza una propuesta de procesamiento, en este caso, asociado a la capacidad de procesamiento de cada uno. Es responsabilidad del agente que busca el servicio determinar de las propuestas cual es la mejor opción.
- Accept_proposal: Una vez se identifica la mejor opción, el agente notifica al agente de la decisión de aceptar la propuesta y envía los datos asociados para su procesamiento.

Dentro del proceso de aceptación y envió de datos a ser procesados, el agente realiza una consulta base al esquema utilizado por el IDS en el cual, se pueden encontrar todos los eventos y detalle de los mismos en una línea de tiempo, al igual que la firma asociada a este.

En la Figura 33, se observa la consulta base para extraer los eventos capturados en la HoneyNet.

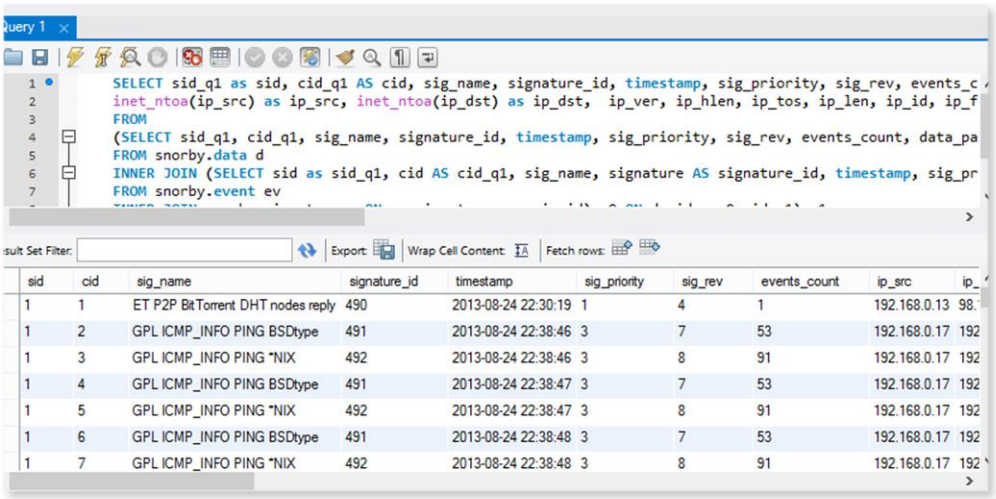

Figura 33. Consulta Eventos Identificados por Snort

Dentro de las columnas más relevantes que podemos encontrar de la consulta anterior están:

- sid: Identificador único dentro de las tablas que permite tener una integridad referencial implícita, siendo que esta no es explícitamente definida como llave foránea en las demás tablas.
- cid: Identificador propio de cada tabla incremental
- sig_name: Nombre del comportamiento identificado que concuerda con la base de datos de firmas asociadas a *Snort*.
- signature_id: Identificador de la firma dentro de la base de datos de *Snort*
- timestamp: Fecha en la cual se identificó el evento
- sig_priority: Prioridad asociada a la firma, es decir, al evento identificado
- sig_rev: Revisión de la firma.

- events_count: Cantidad de veces que se ha presentado el evento
- ip_src: La IP origen desde la cual se generó el evento
- ip_dst: La IP destino a la cual se generó el evento
- data_payload: Data asociada al *payload* que se asocia al evento identificado.

Adicional a los datos anteriores, la consulta devuelve detalles del protocolo IP. El comportamiento general del agente se puede visualizar en la figura 34, en donde se representa el diagrama de secuencia:

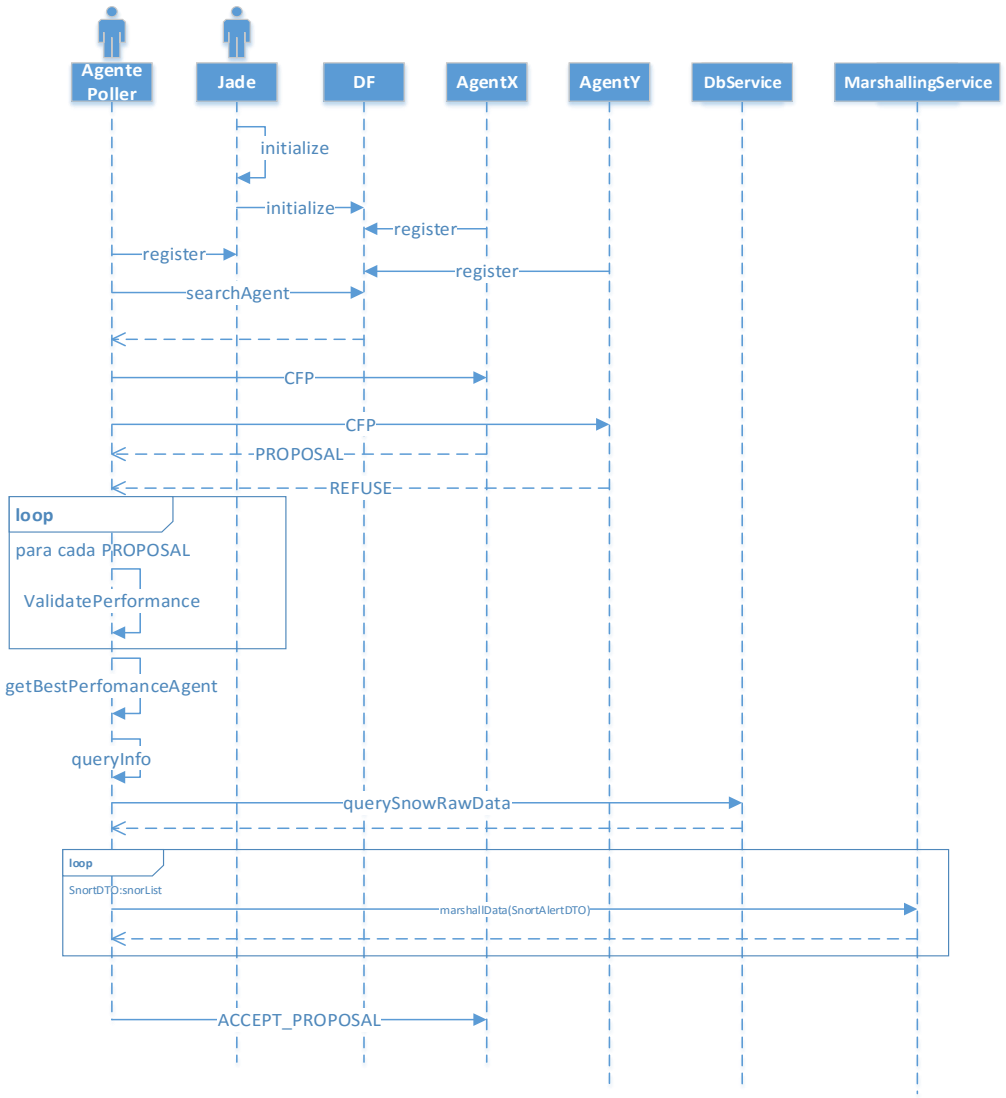

Figura 34. Diagrama de Secuencia Agente Poller

Tabla 10 *Especificación Diagrama Secuencia Agente Poller.*

Origen	Destino	Mensaje	Descripción	Responsable
Jade	Jade	initialize	Inicialización de la plataforma JADE	Capa Media
Jade	DF	initialize	Iniciación del servicio de DF	Capa Media
AgentX	DF	register	Registra del agente Y en el DF, esto es posterior a su registro como tal en la plataforma JADE y RMA	Builder
Agente Poller	Jade	register	Se inicializa registra en JADE	Poller
AgentY	DF	register	Registro del agente Y en el DF, esto es posterior a su registro como tal en la plataforma JADE y RMA	Builder
Agente Poller	DF	searchAgents	Ejecuta el método seach del DFService en búsqueda de agentes bajo el tipo agent-builder	Poller
Agente Poller	Agents(1…n)	send	Envía una petición CFP a los agentes que se encontraron previamente.	Poller
AgentX	Agente Poller	send	El agente al cual se le realiza una petición de inicio, responde con un PROPOSAL, es decir, una propuesta de trabajo.	Buillder
AgentY	Agente Poller	send	El agente de acuerdo a su criterio (Dado por su comportamiento) define que no puede o no está disponible para aceptar la propuesta, por lo cual retorna un REFUSE.	Builder
Agente Poller	Agente Poller	validatePerformance	El agente Poller válida para cada propuesta cual ofrece el mejor rendimiento en el procesamiento de datos.	Agente Poller
Agente Poller	Agente Poller	getBestPerfromanceAgent	Se obtiene, después de ser procesados, el mejor agente acorde a la propuesta realizada.	Agente Poller
	Agente Poller	queryInfo	Se consultan los eventos asociados al IDS en la base de datos de Snort.	Agente Poller

Agente Poller				
Agente Poller	DBService	querySnortDataRaw	Se utiliza el servicio de base de datos para consultar los eventos de Snort.	Agente Poller
Agente Poller	Marshalling Service	marshallData	Se utiliza el servicio de Marshalling para procesar los eventos y darles un formato específico.	Agente Poller
Agente Poller	AgentX	send	Se envía la data a ser procesada y formateada al agente Builder que ofreció el mejor servicio acorde a los parámetros definidos.	Agente Poller

Los datos después de ser procesados dentro del Agente *Poller* para ser enviados al Agente *Builder* que realizó la mejor propuesta, son serializados o formateados en JSON para ser enviados. En la figura 35, se puede observar los eventos capturados en formato correspondiente durante su envío a el agente *builder*.


```
    {"sid":1,"cid":292,"sigName":"ET CURRENT_EVENTS Exploit Kit Delivering
  Compressed Flash Content to Client","signatureId":512,"timestamp":"ago 25,
2013","sigPriority":2,"sigRev":1,"eventsCount":21,"ipSrc":"201.232.123.9","ipDs
t":"192.168.0.13","ipVer":4,"ipHlen":5,"ipTos":0,"ipLen":1500,"ipId":22051,"ipF
           lags":0,"ipOff":0,"ipTtl":56,"ipProto":6,"ipCsum":57681
```


Figura 35. Eventos de ataques en formato de envío

Este agente puede ser ejecutado en diferentes componentes dentro de la red trampa señuelo, o inclusive, en diferentes locaciones geográficas, teniendo en cuenta que debe de tener conectividad hacia el esquema de *Snort* y a la plataforma JADE. Para nuestro caso de ejemplo y para validar el correcto funcionamiento del Agente vamos a realizar una prueba sencilla de ejecución y visualización dentro del RMA. La ejecución del componente de capa media JADE se realiza través de la utilidad de consola del sistema operativo, la del agente se realiza a través de una configuración

específica dentro del entorno de desarrollo Eclipse. En la figura 36, se muestra el agente *poller* en ejecución.

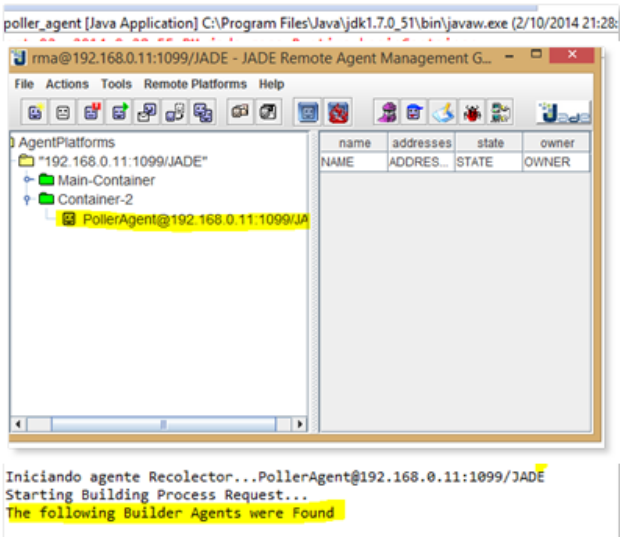

Figura 36. Ejecución Agente Poller RMA Jade

- **Agente Builder:** El agente *Builder* nace de las mismas necesidades expuestas durante la implementación del agente *Poller*, la diferencia en este caso, es el dominio en el cual se contextualiza el agente, al igual que las responsabilidades asignadas al mismo.

  Este componente tiene asignada la responsabilidad de recibir y procesar los datos o eventos que un elemento externo a él capture, en este caso, un agente. Dentro de su flujo principal debe de garantizar que se registre dentro del DF de JADE y que tipifique su servicio como "*building-service*", una vez realiza este objetivo, se le definen dos comportamientos cíclicos secuenciales, el primero para identificar una propuesta inicial de trabajo, en cuyo caso y acorde a las variables que este tienen definidas para determinar su rendimiento en el momento de la petición, responde con una oferta inicial. El segundo comportamiento se encarga de obtener la

respuesta del consumidor del servicio y construir a partir de los datos enviados los casos base de conocimiento para su posterior uso dentro de componente de razonamiento. Al finalizar, se encarga de notificar al agente deliberativo de la existencia de nuevos datos.

En la figura 37, se observa el flujo asociado a la ejecución puede ser visualizado a continuación:

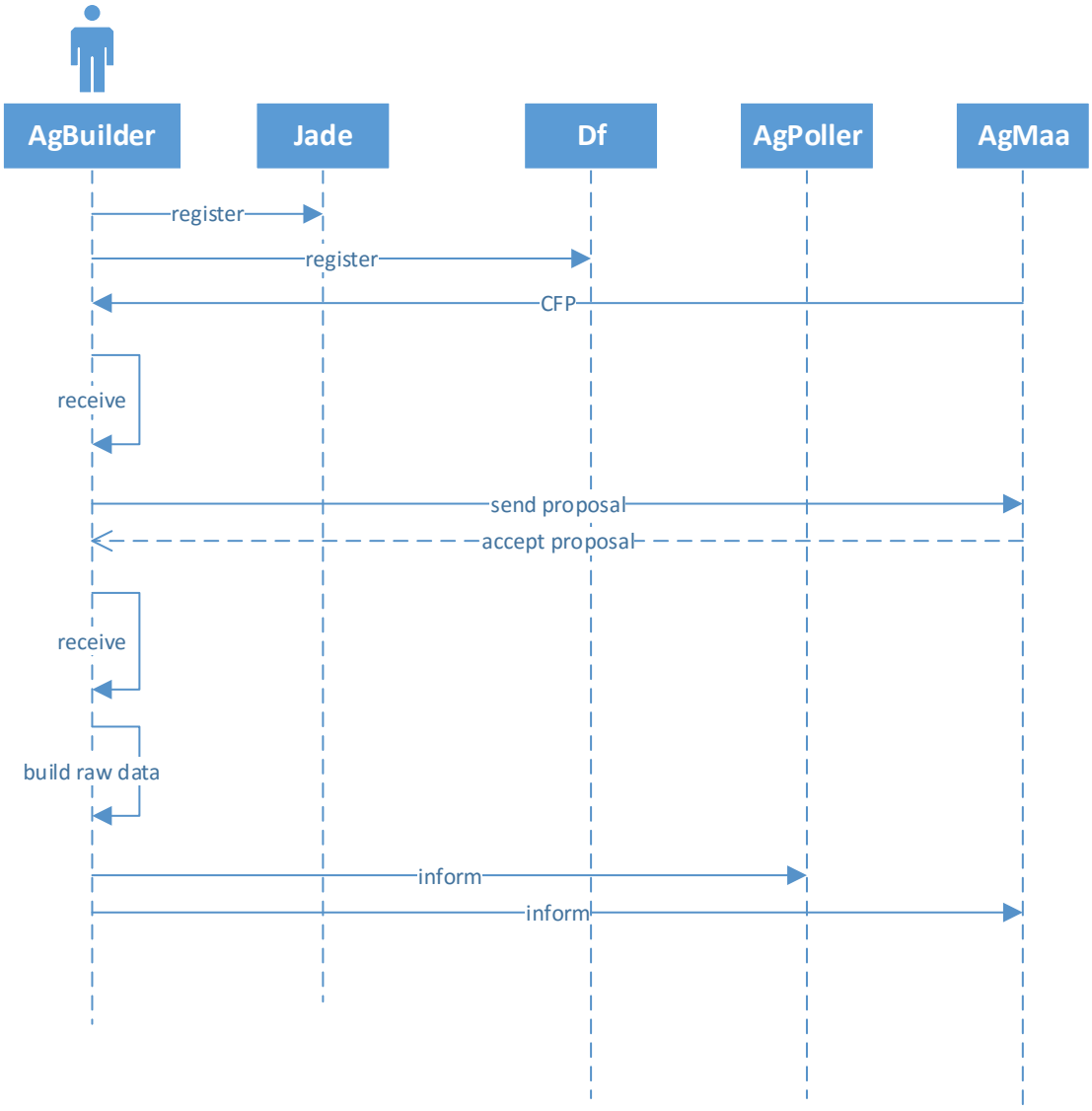

Figura 37. Diagrama de Secuencia Agente Builder – RMA

Tabla 11 *Especificación Diagrama Secuencia Agente Builder –RMA.*

Origen	Destino	Mensaje	Descripción	Responsable
Agente Builder	Jade	Register	Registro en la plataforma Jade	Capa Media
Agente Builder	DF	register	Registro en el DF	Capa Media
Agente Poller	Agente Builder	send	Se recibe una petición por parte del agente Poller CFP para iniciar la actividad ofrecida por el agente.	Builder
Agente Builder	Agente Poller	send	Una vez se recibe la petición se calcula la capacidad actual y se envía la respuesta al Agente mediante un PROPOSE.	Poller
Agente Builder	Agente Poller	buildRawDat a	Se verifica si la respuesta es de aceptación y se obtiene los datos asociados a la solicitud, se procesan los datos y se almacenan en la base de conocimiento local.	Builder
Agente Poller	AgentX	send	Se envía la data a ser procesada y formateada al agente Builder que ofreció el mejor servicio acorde a los parámetros definidos.	Agente Poller

Una vez se reciben los mensajes estos quedan almacenados en una base de datos diferente, definidos por el siguiente diagrama de dominio, representado en la figura 38.

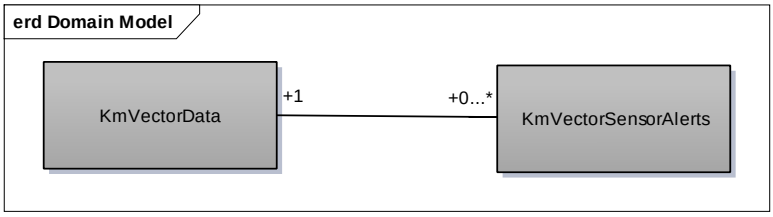

Figura 38. Modelo de Dominio Casos

Las dos entidades que se muestran en el diagrama muestran de la manera más resumida lo que define la base de conocimiento para el componente deliberativo.

Es importante tener en cuenta que ambas, definen la descripción y solución a un problema, el de encontrar patrones de ataque en la base de conocimiento. Adicional a la persistencia de los casos, se define una tabla lo menos relacional posible asociada a la extracción de datos hecha previamente por el agente *poller*, esto con el objetivo de facilitar la aplicabilidad de diferentes técnicas de minería de datos y/o procesos de extracción, transformación y limpieza para que faciliten su definición como una posible fuente de datos para un *datamart*[36] o cubo de datos. En la figura 39, muestra la estructura de la tabla de eventos del sistema.

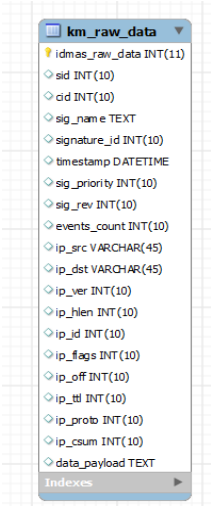

*Figura 39.*Tabla de datos sin procesar de eventos

Para validar el funcionamiento del agente se va a realizar la ejecución aislada y unitaria, visualizando el resultado inicial y la carga en el RMA de JADE. En la figura 40, se muestra el agente *builder* en ejecución.

[36] Subconjunto de datos correspondiente a un dominio especifico y sirven como fuente de datos para un Cubo.

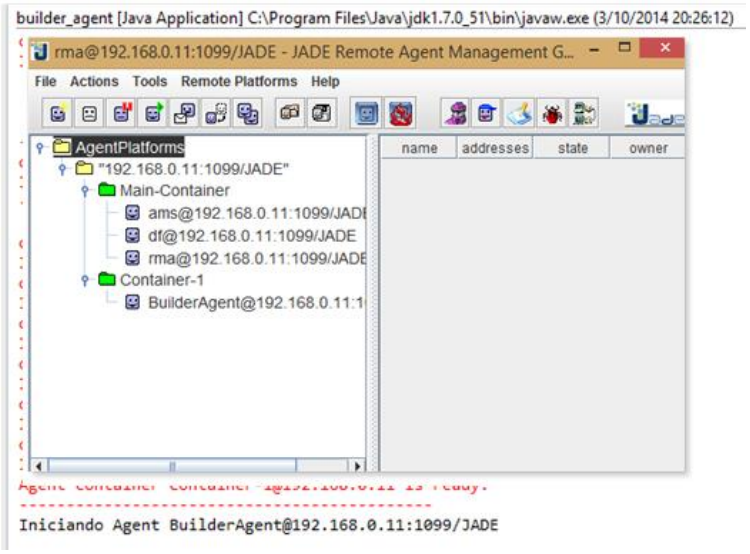

Figura 40. Agente Builder Jade RMA

- **Agente Maa – UI Deliberativo:** El agente Deliberativo es como tal el encargado de permitir la configuración previa para realizar la ejecución del CBR, adicional, se encarga de ejecutar y persistir los nuevos vectores de ataques que se encuentran. Su funcionamiento base parte de la notificación de la existencia de nueva información a ser procesada por parte del agente *builder*, una vez esta es recibida, se lanza la consulta de clasificación KNN sobre la base de conocimiento, esto para obtener los 5 primeros casos de coincidencia y persistirlos. La decisión de si estos son válidos o no están sujetas a un experto y deben de ser realizadas inicialmente en la BD.

El flujo asociado puede ser visto a continuación en la figura 41.

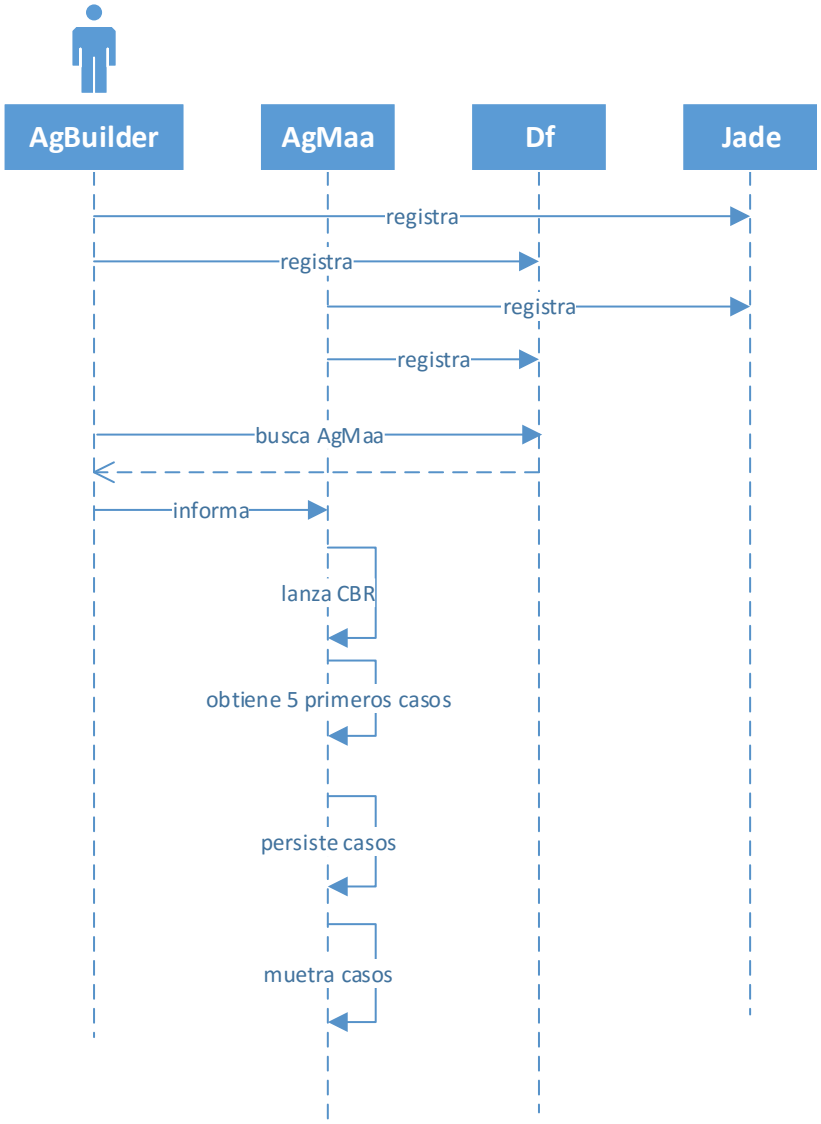

Figura 41. Diagrama de Secuencia Agente Maa – Deliberativo

Tabla 12 *Especificación Diagrama de Secuencia Agente Maa – Deliberativo.*

Origen	Destino	Mensaje	Descripción	Responsable
Agente Builder	Jade	Register	Registro en la plataforma Jade	Capa Media
Agente Builder	DF	register	Registro en el DF	Capa Media
Agente Maa	Jade	Register	Registro en la plataforma Jade	Capa Media

Agente Maa	DF	Register	Registro en la plataforma Jade	Capa Media
Agente Builder	DF	buscarMaa	Busca en el DF un agente Maa para informar sobre nueva Data a ser procesada	Capa Media
Agente Builder	AgenteMaa	inform	Informa al agente Maa deliberativo sobre nueva data a ser procesada.	Agente Builder
Agente Maa	AgenteMaa	Lanza CBR	Se encarga de lanzar el CBR con la configuración previa y posterior a la notificación de nueva Data.	AgenteMaa
Agente Maa	AgenteMaa	Obtener 5 primeros casos	Se encarga de obtener los 5 primeros casos acorde a la búsqueda previa.	AgenteMaa
Agente Maa	AgenteMaa	Persistir Casos	Persisten los casos obtenidos para su revisión.	AgenteMaa
Agente Maa	AgenteMaa	Mostrar Casos	Actualiza los casos para ser visualizados en el Front	AgenteMaa

Para validar el funcionamiento se realiza la ejecución del agente en conjunto con el sistema para poder visualizarlo de manera correcta. En la figura 42, se muestra el sistema multi-agente en ejecución.

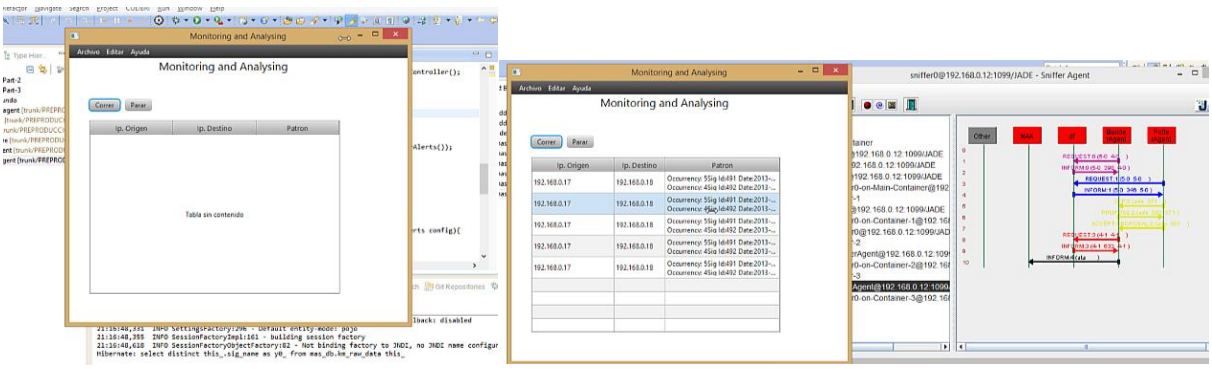

Figura 42. Sistema Multi-Agente Deliberativo

Con base a los resultados obtenidos previamente, se realiza la ejecución de los agentes asociados al proceso de extracción y procesamiento de los datos en su formato puro. Posterior a esto son transformados para ser persistidos para la base de conocimiento. En la figura 43, se puede ver el comportamiento base de interacción entre los agentes.

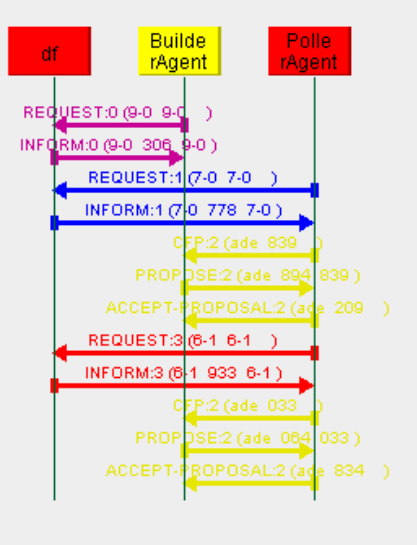

Figura 43. Agente Sniffer

Inicialmente se realiza el registro del agente *Builder* ante el DF bajo la siguiente petición. En la figura 44, se muestra el envío de un mensaje tipo *request* bajo el estándar FIPA ACL.

```
((action
    (agent-identifier
      :name df@192.168.0.13:1099/JADE
      :addresses (sequence http://family-pc:7778/acc))
    (register
      (df-agent-description
        :name
          (agent-identifier
            :name BuilderAgent@192.168.0.13:1099/JADE
            :addresses (sequence http://family-pc:7778/acc))
        :services
          (set
            (service-description
              :name Builder-Agent
              :type builder-agent))))))
```

Figura 44. Envío de mensaje (Request FIPA ACL)

Posterior a esto se realizan los siguientes pasos dentro de la interacción definida, en donde en la penúltima petición, se realiza el envío de los datos obtenidos y procesados en su fase inicial por el agente *poller*. En la figura 45, se muestra el ejemplo de los eventos capturados y el formato en el cual son enviados entre agentes.

{"sid":1,"cid":1,"sigName":"ET P2P BitTorrent DHT nodes reply","signatureId":490,"timestamp":"ago 24, 2013","sigPriority":1,"sigRev":4,"eventsCount":1,"ipSrc":"192.168
ntsCount":53,"ipSrc":"192.168.0.17","ipDst":"192.168.0.18","ipVer":4,"ipHlen":5,"ipTos":0,"ipLen":84,"ipId":0,"ipFlags":0,"ipOff":0,"ipTtl":64,"ipProto":1,"ipCsum":474
"ipCsum":47413,"dataPayload":"F135195207E80900080900A0B0C0D0E0F101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637"}split{"sid":1,"cid":5,"
:1,"cid":7,"sigName":"GPL ICMP_INFO PING *NIX","signatureId":492,"timestamp":"ago 24, 2013","sigPriority":3,"sigRev":8,"eventsCount":91,"ipSrc":"192.168.0.17","ipDst":
17","ipDst":"192.168.0.18","ipVer":4,"ipHlen":5,"ipTos":0,"ipLen":84,"ipId":0,"ipFlags":0,"ipOff":0,"ipTtl":64,"ipProto":1,"ipCsum":47413,"dataPayload":"F335195202E409

Figura 45. JSON Eventos Capturados y Procesados

Una vez se reciben estos datos son procesados y persistidos en base de datos acorde al esquema planteado previamente. El flujo anterior es el que se sigue normalmente dentro de la ejecución de la plataforma, sin límite alguno en el registro de cualquiera de los agentes, siempre y cuando su nombre sea único dentro del contenedor principal.

4. Comportamiento deliberativo en el componente CBR: Dentro del planteamiento inicial y para garantizar un comportamiento deliberativo dentro de la plataforma se realiza la implementación de una de las estrategias más comunes y fundamentales dentro del dominio de la inteligencia artificial; el razonamiento basado en casos, que utiliza, sin

basarse únicamente en conocimiento general de un dominio o realizar asociación entre descriptores y conclusiones, experiencias obtenidas previamente, que se definen como casos. Su comportamiento básico se fundamente en la búsqueda de casos previos que cumplan ciertos criterios de similitud y que se pueda reutilizar. A continuación se presenta un modelamiento básico de la las actividades asociadas al proceso de CBR. En la figura 46, Aamodt, A., & Plaza, E. (1994) muestra el desglose de tareas de un CBR.

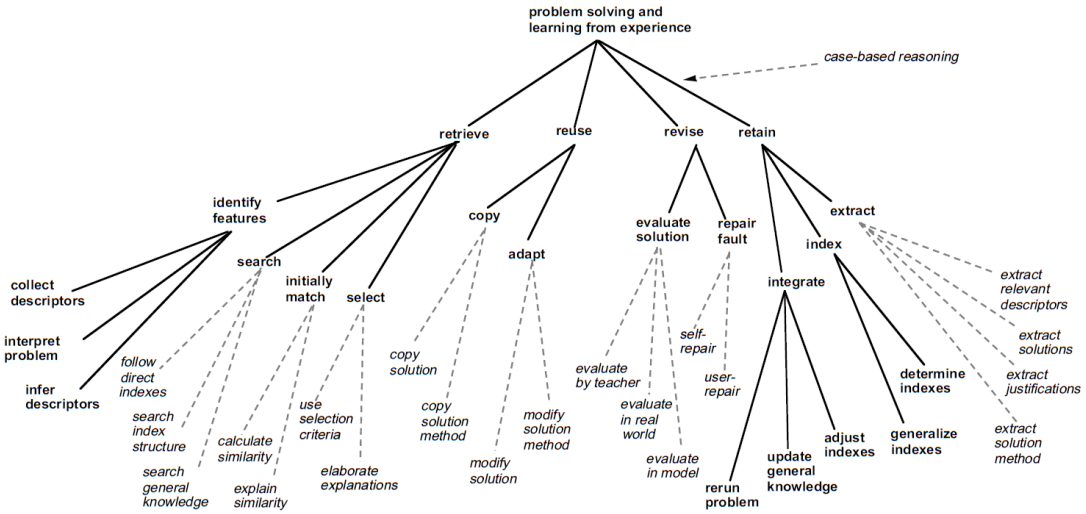

Figura 46. Aamodt, A., & Plaza, E. (1994) Descomposición basada en tareas de un CBR

De manera general las fases asociadas pueden ser descritas de la siguiente forma:

a. RETRIEVE u obtención, esta fase se encarga de obtener casos previos de la base de conocimiento, acorde a la configuración base realizada. En este punto la fuente de datos ha sido el esquema propio del IDS, el cual fue consultado por el agente extractor y entregado para su procesamiento a un agente constructor.

b. REUSE o reutilización, con base a la información obtenida se revisa y plantea una propuesta de solución al problema dado. Dentro de la obtención de los resultados se realizó una configuración base de parámetros de peso y similitud para su comparación con la base de conocimiento obtenida del IDS, esto da como resultado un posible escenario que resuelve el problema planteado dentro de la búsqueda de nuevos comportamientos o vectores de ataque.
c. REVISE o revisión, se realiza una aprobación del planteamiento de solución. En este punto los casos obtenidos y acordes al planteamiento de la hipótesis del proyecto, son aprobados de manera automática, no siendo esto una condición de fondo y pudiéndose crecer en la revisión.
d. RETAIN o retención, se almacenan los casos que se consideran útiles.

Dentro de la implementación técnica de la solución se utiliza JColibri como librería base que brinda las utilidades necesarias para apoyar al CBR en cada una de sus fases. En la figura 47, (Recio-García, Díaz-Agudelo & González-Calero, 2008) se observa la arquitectura utilizada por la herramienta.

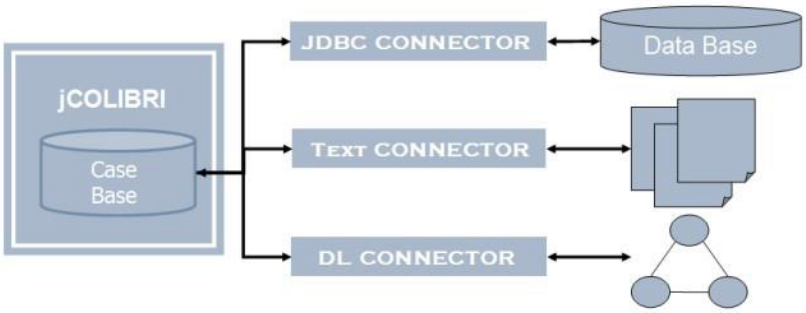

Figura 47. Arquitectura Base JColibri. (Recio-García, Díaz-Agudelo & González-Calero, 2008)

En la figura 48, se observa la estructura base sobre la cual se realiza el desarrollo es la siguiente:

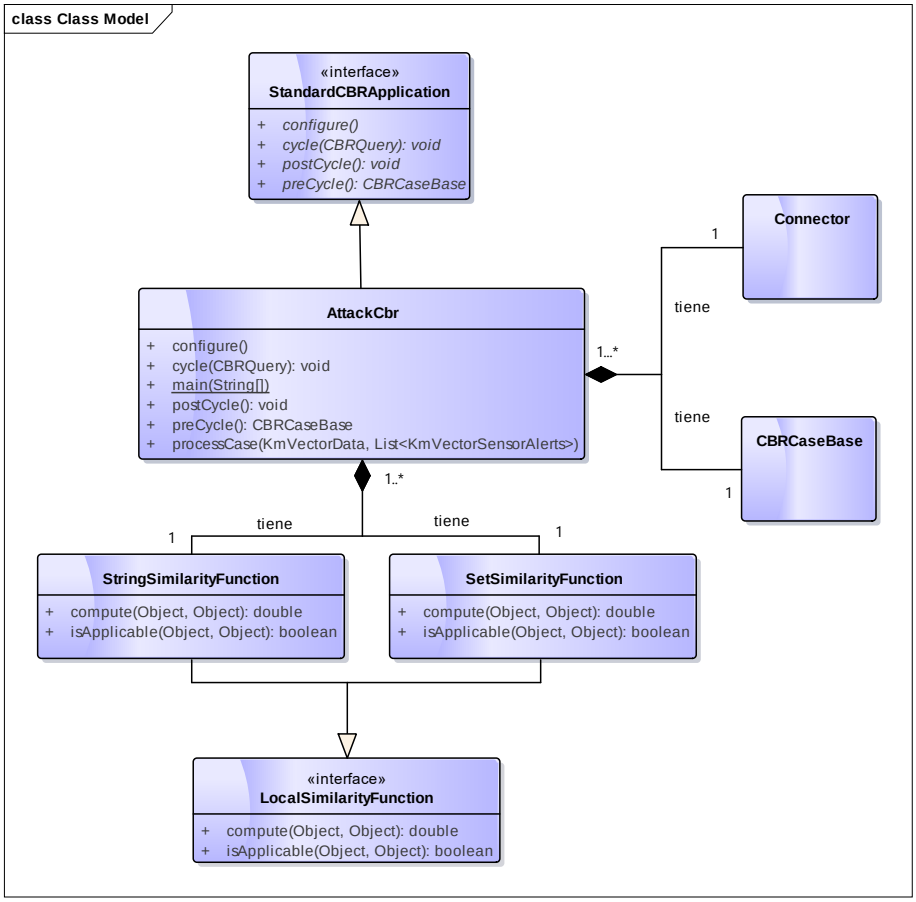

Figura 48. Diagrama de Clases, del componente CBR

En la anterior especificación se pueden visualizar los componentes claves dentro de la definición de una aplicación CBR en el marco JColibri, es importante resaltar la implementación de los siguientes métodos ya que estos definen el ciclo de vida de la aplicación:

- *Configure*: Se realiza a inicialización de clases, similar al constructor.
- *Precycle*: Realiza la inicialización de la aplicación, este punto es muy útil en caso de ser necearía la ejecución de algoritmos complejos o que tengan un alto consumo de recursos, esto porque solo se ejecuta una vez. En este punto lo que se

hace es cargar la configuración asociada a base de datos y la consulta inicial de los casos de la línea base.

- *Cycle*: Ejecuta el proceso u objetivo del CBR, su ejecución es reiterativa. Para este caso no se realiza ninguna operación durante la implementación.
- *Postcycle*: Se ejecuta al final del ciclo de vida, usualmente útil para dar de baja a recursos y liberar conexiones.

Adicional y como comportamiento propio de la plataforma se adiciona un método cuyo objetivo es ejecutar las fases de reutilización, revisión y almacenamiento de los nuevos casos.

Dentro de este, se resalta la configuración de las funciones de similitud para poder realizar la categorización y búsqueda de nuevos casos, asociadas directamente al algoritmo KNN. Este basa su comportamiento en la identificación y categorización de los elementos contra los que se encuentran a su alrededor y en su base de comparación K. En la figura 49, se muestra la estrategia base del algoritmo KNN.

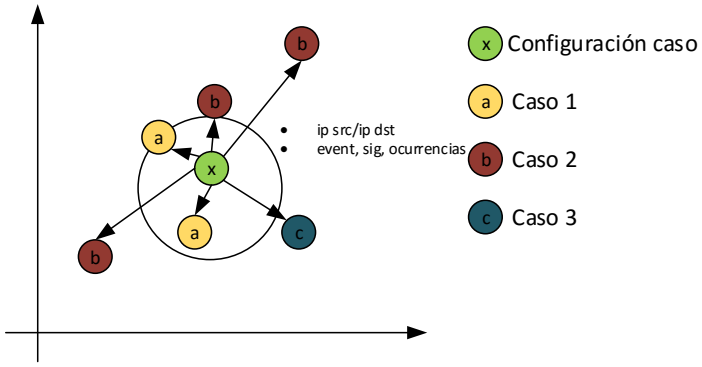

Figura 49. Algoritmo KNN de Clasificación

Del comportamiento que se muestra en la gráfica se pude definir que la configuración x definida previamente, se encuentra acorde a la proximidad cerca de los elementos a, b y c, siendo estas posibles tipificaciones para esta, sin embargo y debido a la cantidad de elementos cercanos se clasifica como (**a**) La configuración que se usa para la propuesta se basa en los siguientes elementos, definidos previamente en el modelo de dominio:

- Ip Origen
- Ip Destino
- Alertas
    - Identificación de alerta
    - Ocurrencia
    - Fecha

La estructura base que soporta el modelo de dominio se puede apreciar en la figura 50 es:

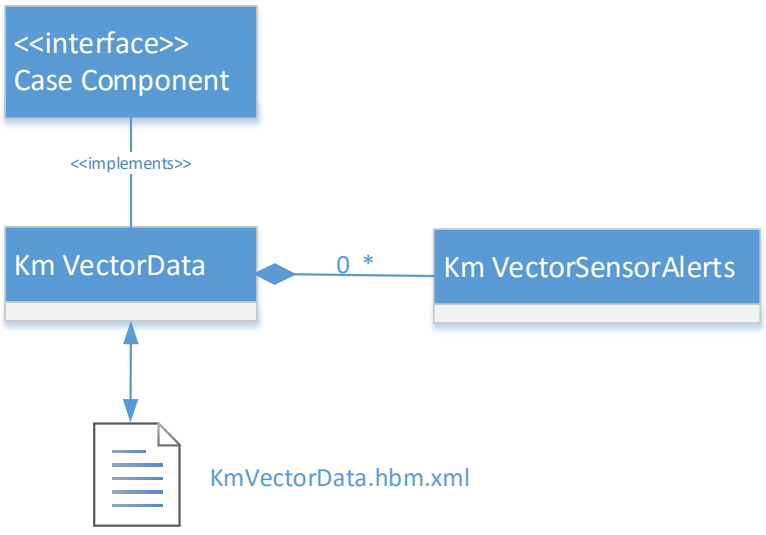

Figura 50. Estructura CBR para la Plataforma

- *Configuración de criterios de selección*

La configuración base surge de comprender la estructura que soporta tanto la captura de los eventos, la figura 51, presenta un ejemplo sencillo de la interacción a nivel de componentes dentro del entorno virtual vulnerable.

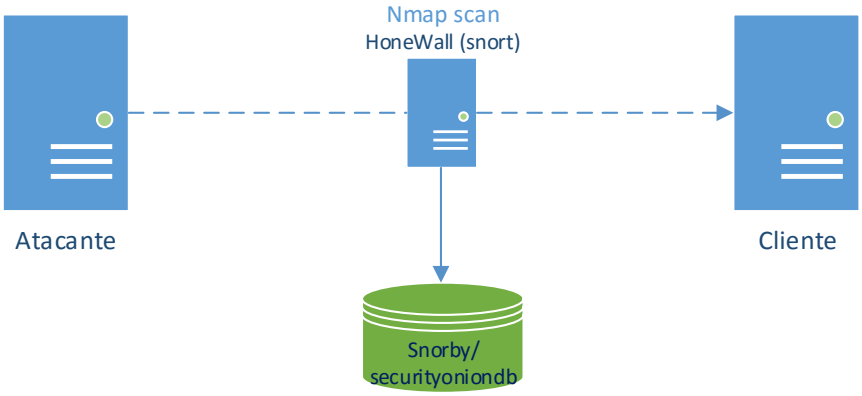

Figura 51. Ejemplo de Entorno Vulnerable

Una vez se realiza la ejecución se puede visualizar el resultado en el *sguil*, en la figura 52.

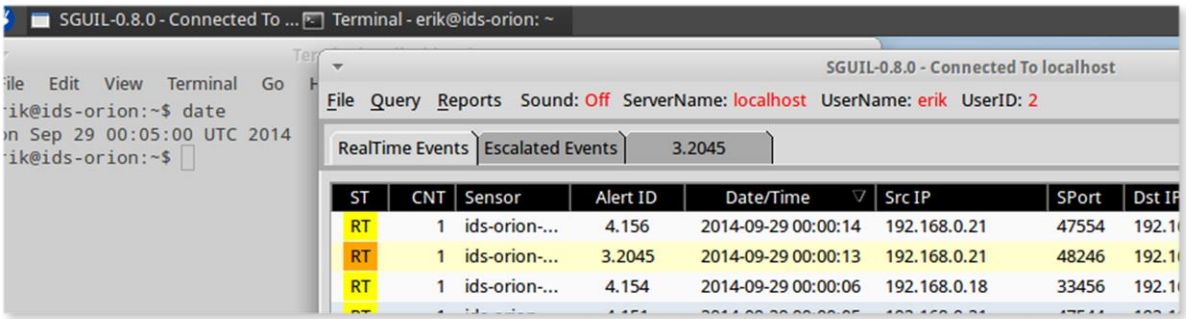

Figura 52. SGUIL Visualización de Alertas

Los criterios de selección permiten determinar de la base de conocimiento previamente generada y actualizada, cuales nuevos patrones de posibles ataques pueden ser encontrados.

En este caso se configuran los siguientes atributos, pudiéndose ampliar según sea la necesidad:

Tabla 13 *Configuración de Criterios de Selección.*

Elemento	Tipo	Peso	Descripción
IP Origen	Similitud	1	IP origen desde la cual se lanzó el posible ataque
IP destino	Similitud	1	IP destino a la cual se lanzó el posible ataque
Alertas Asociadas	Similitud	1	Conjunto de alertas asociadas que incluyen la firma, fecha y ocurrencias.

La configuración asociada se realiza a través de la interfaz gráfica de la aplicación, teniendo en cuenta que el peso se define por defecto en 1, siendo posible su modificación. En la figura 53, muestra la interfaz gráfica del sistema multi-agente.

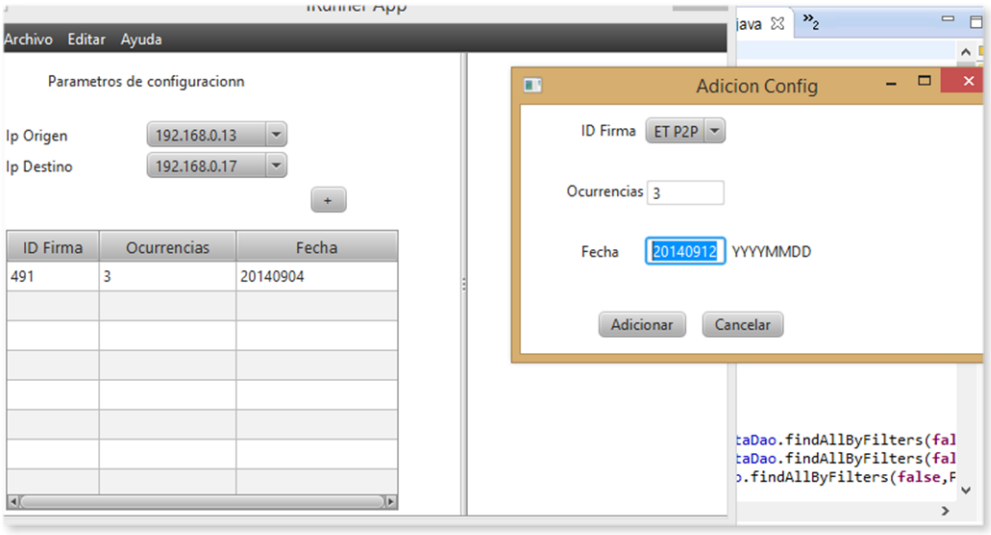

Figura 53. Front-End Plataforma

La configuración de estos criterios dentro de JColibri, se genera mediante los siguientes pasos:

- Se debe de realizar la carga de la configuración dentro de la descripción del CBRCase.
- Se realizan las configuraciones asociadas a las variables y funciones de comparación.
- Se realiza la evaluación correspondiente de similitud.

Por defecto, JColibri no ofrece una función de similitud para *Sets*[37], por lo cual fue necesario realizar el siguiente planteamiento:

$$Sea\ A = \{Alerta\ 1, Alerta\ 2, \ldots., Alerta\ n\}$$

[37] Tipo de Arreglo de Java.

$$Sea\ B = \{Alerta\ 1, Alerta\ 2, \dots Alerta\ n$$

En donde A son las alertas asociadas a patrones de ataques ya identificados y B un nuevo patrón configurado:

$$A\ \cap B = \{Alerta\ 1, Alerta\ 3\}$$

La intersección de los conjuntos define los elementos comunes a ambos, por consiguiente, si n es el total de elementos del conjunto A y m es el total de elementos del conjunto resultante de la intersección:

$$SV = \sim \frac{\left(\left(\frac{n}{m}\right) * 100.0\right)}{100}$$

Siendo SV el resultado de similitud y realizando una aproximación sobre el valor del resultado.

La función de comparación de cadenas que JColibri utiliza se basa en el método *equals* asociado a Java, el cual basa su funcionamiento en el siguiente planteamiento:

$$Sea\ A = cadena1$$

$$Sea\ B = cadena2$$

$$Sea\ C = \{c, a, d, e, n, a, 1\}\ el\ conjunto\ de\ la\ cadena\ A$$

$$Sea\ D = \{c, a, d, e, n, a, 2\}\ el\ conjunto\ de\ la\ cadena\ B$$

$$Si\ C\ \cap D = \emptyset, entonces\ C\ \neq D$$

- Front-end de la aplicación

El componente gráfico que apoya la solución es una implementación básica en java utilizando *JavaFX*. Los elementos principales dentro de esta se podrían definir en los siguientes:

- El desarrollo de la maqueta se específica en fxml utilizando la herramienta A nivel de interfaz se encuentran los siguientes componentes:
- Consulta de IP´s: La consulta sirve como base para definir los criterios de configuración, esta se basa en los datos capturados por *Snor*t. Estos datos sirven tanto para el origen como para el destino.

En la figura 54, se muestra la interfaz de configuración base para la ejecución del CBR.

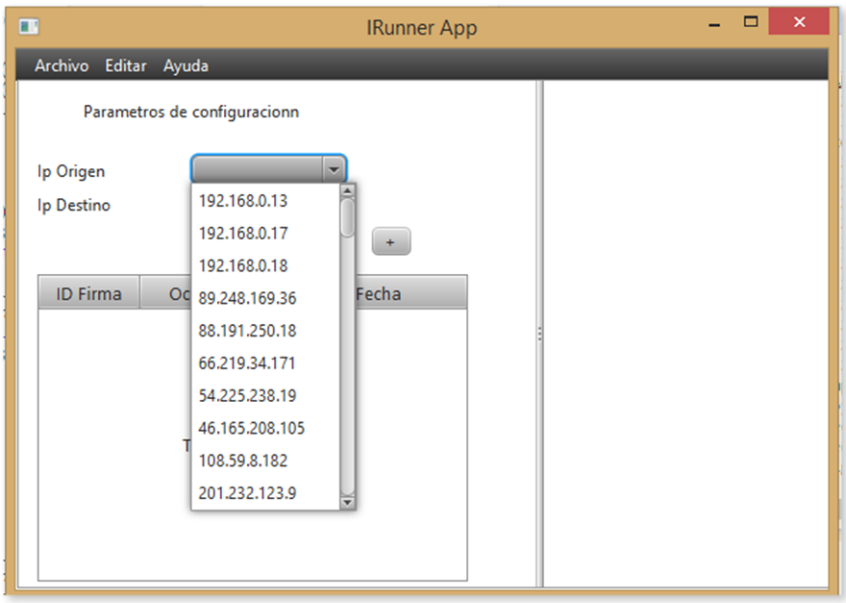

Figura 54. Front-End Plataforma Configuración de Búsqueda

- La configuración de los patrones en donde incluye la alerta, la ocurrencia y la fecha especificada. Estos datos se basan en la base de datos de firmas identificada por *snort*.

En la figura 55, se muestra la selección de firmas de ataques durante la configuración base del CBR.

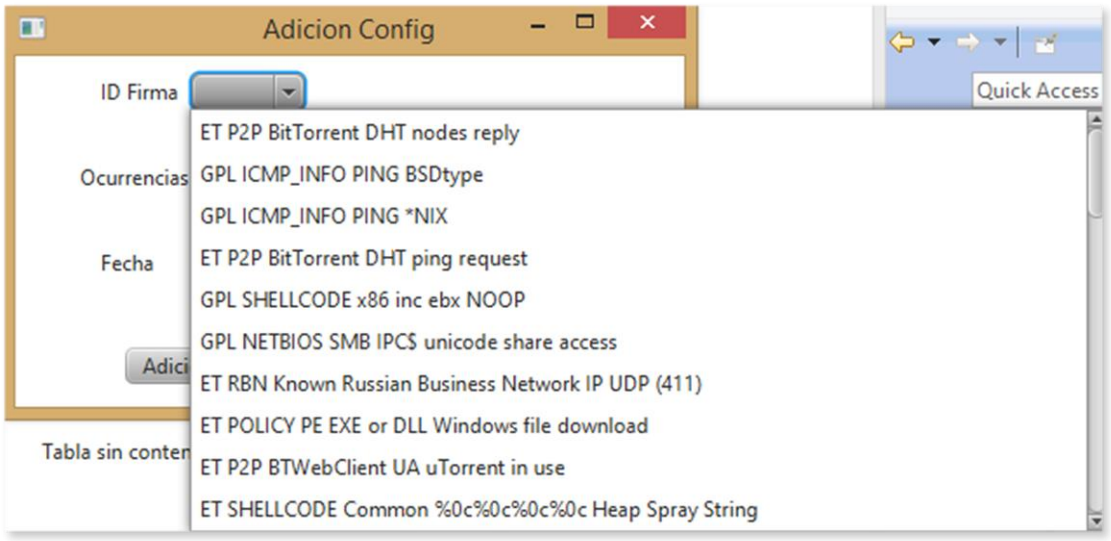

Figura 55. Front-End Plataforma Configuración de Búsqueda 2

3.2.2.3 Pruebas Integrales.

Para realizar las pruebas de manera exitosa se deben establecer las siguientes estructuras de datos:

- Conjunto de datos base referentes a las diferentes alarmas que sean capturadas de los diferentes despliegues de los sensores *HoneyNet*. Este conjunto de datos son obtenidos directamente de la base de datos por un agente específico y contiene la siguiente información mostrada en la figura 56.

Table: km_raw_data

Columns:

idmas_raw_data	int(11) AI PK
sid	int(10)
cid	int(10)
sig_name	text
signature_id	int(10)
timestamp	datetime
sig_priority	int(10)
sig_rev	int(10)
events_count	int(10)
ip_src	varchar(45)
ip_dst	varchar(45)
ip_ver	int(10)
ip_hlen	int(10)
ip_id	int(10)
ip_flags	int(10)
ip_off	int(10)
ip_ttl	int(10)
ip_proto	int(10)
ip_csum	int(10)
data_payload	text

Figura 56. Estructura de tabla base de Alertas

- Estructura base de los casos contra los cuales se va a llevar a cabo la extracción de posibles casos con el CBR. Esta información es procesada y almacenada a partir de los datos de las alarmas capturadas en primera instancia. En la figura 57, se presenta la estructura de datos asociadas a los eventos capturados y a su procesamiento.

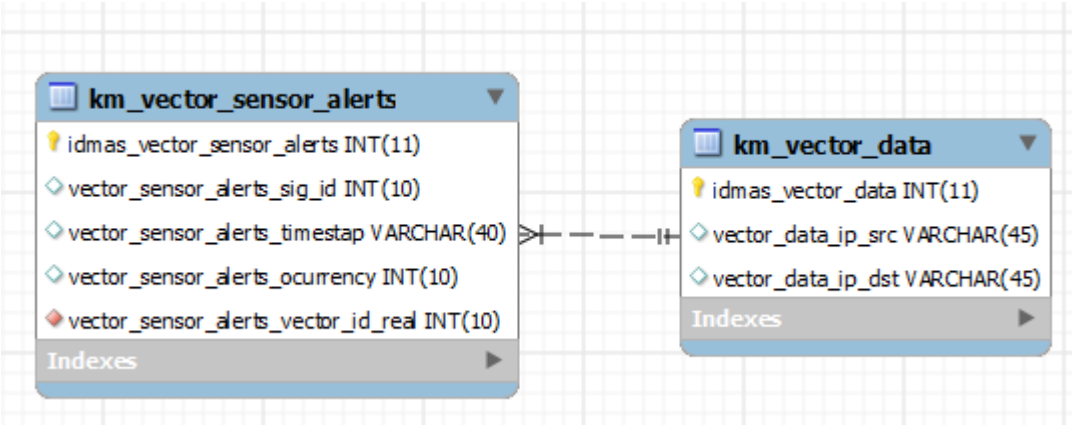

Figura 57. Estructura de tabla casos base

- Reglas asociadas y afines a la estructura de casos base que condicionan la búsqueda inicial y base el CBR. En la figura 58, se muestra la configuración base de reglas que podrían ser utilizadas en la ejecución del CBR.

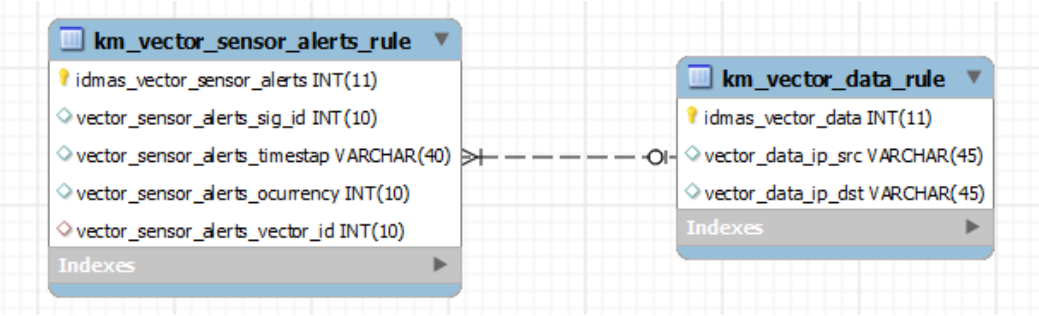

Figura 58. Reglas asociadas a la estructura de casos base

Los escenarios base y con los cuales se define un punto de partida son los siguientes:

Tabla 14 *Configuración Base de Búsqueda CBR.*

Casos					
Elemento	**Operador**	**Valor**			**Diagnostico**
Ip_src	=	x			
dst_src	=	Z			Posible descubrimiento de información de X
sig_id	=	365	366	469	
timestamp	between	current-10	current		
events_count	>	10	10	10	

Ip_src	=	X			
dst_src	=	Z			
sig_id	=	718	492	716	Posible intrusión a través de Telnet de X
timestamp	between	current-10	current		
events_count	>	3	3	1	

Los casos anteriormente mencionados son un punto de partida ya que pueden ser adicionados nuevos casos a ser evaluados acorde a lo encontrado durante la ejecución del sistema multi-agente deliberativo.

3.2.2.3.1 Ejecución completa del sistema MAS+CBR.

1. Verificación de base de datos local: Se verifica la tabla en la base de datos de la plataforma para garantizar que inicialmente no haya ningún tipo de datos. En la figura 59, se puede apreciar la tabla base de eventos vacía sin procesar, antes de la ejecución.

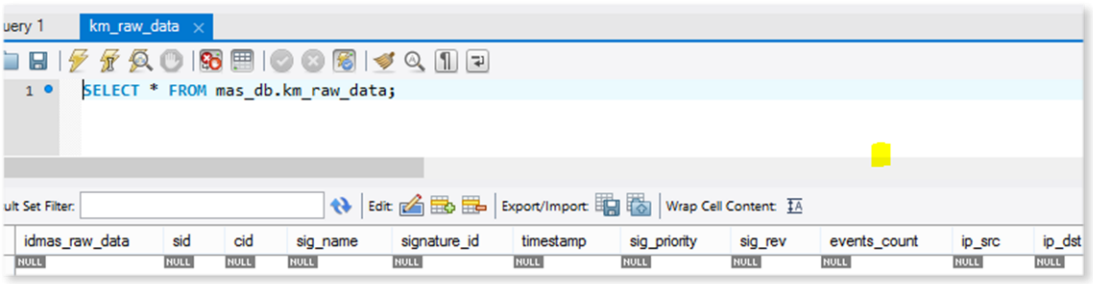

Figura 59. Consulta Base de Conocimiento de Ataques

En la figura 60, se muestra la tabla base de eventos procesada, vacía.

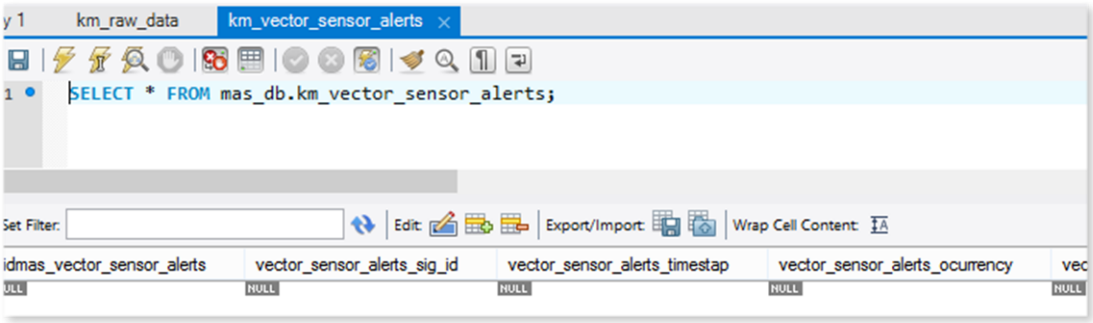

Figura 60. Consulta Base de Conocimiento de Ataques 2

2. Ejecución agentes: Inicialmente se inicia el agente *Builder*, para ofrecer soporte de construcción y procesamiento de datos. En la figura 61, se observa la inicialización del agente *builder*.

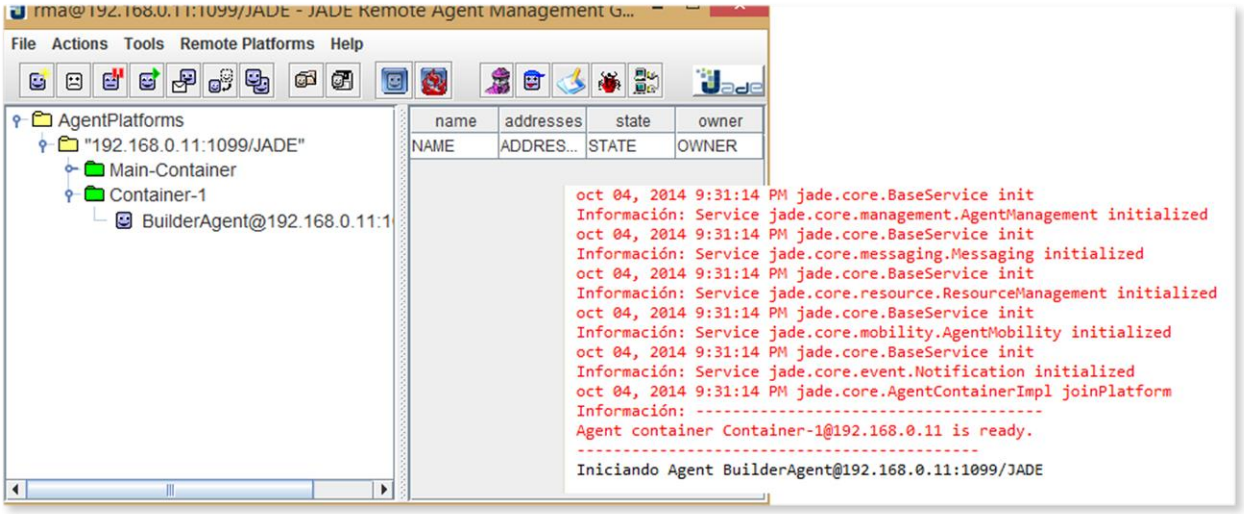

Figura 61. Ejecución Agente builder Prueba Integral

Posterior, se inicia el agente *Poller* para que se encargue de cargar los posibles ataques que se generaron dentro de *SecurityOnion*, con *Snort*. En la figura 62, se aprecia la inicialización del agente *Poller*.

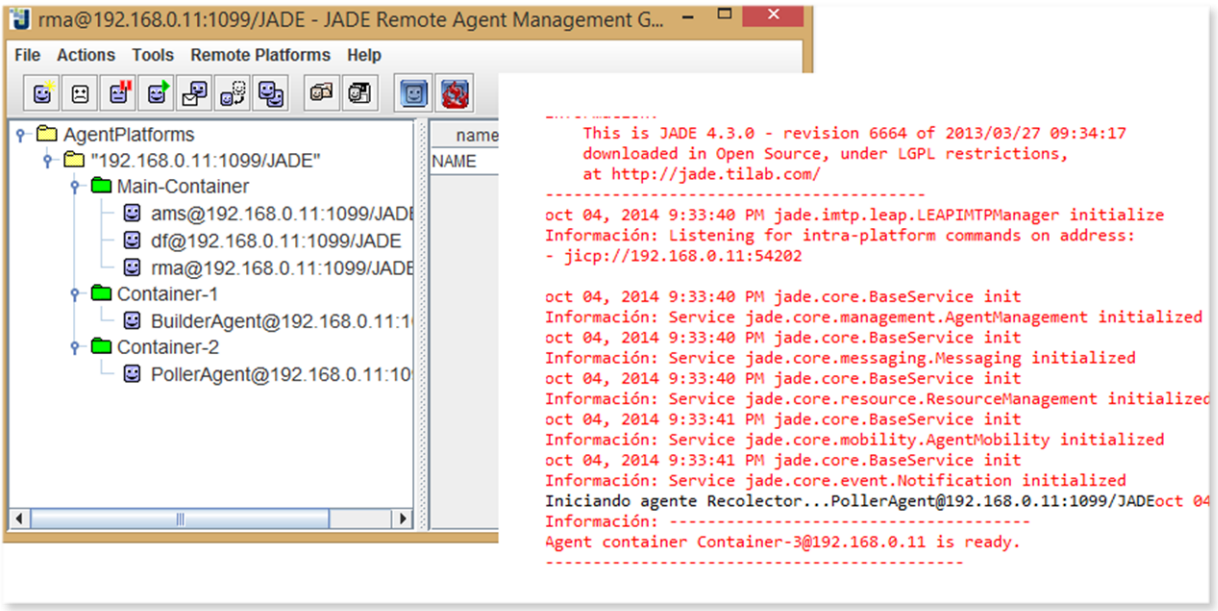

Figura 62. Ejecución Agente Poller, Prueba Integral

Al revisar el *log* podemos identificar que se encontró un agente y que el agente *Builder* registrado recibió la petición y se llevó a cabo de manera exitosa. En la figura 63, se muestra la recepción de datos por parte del agente *builder*.

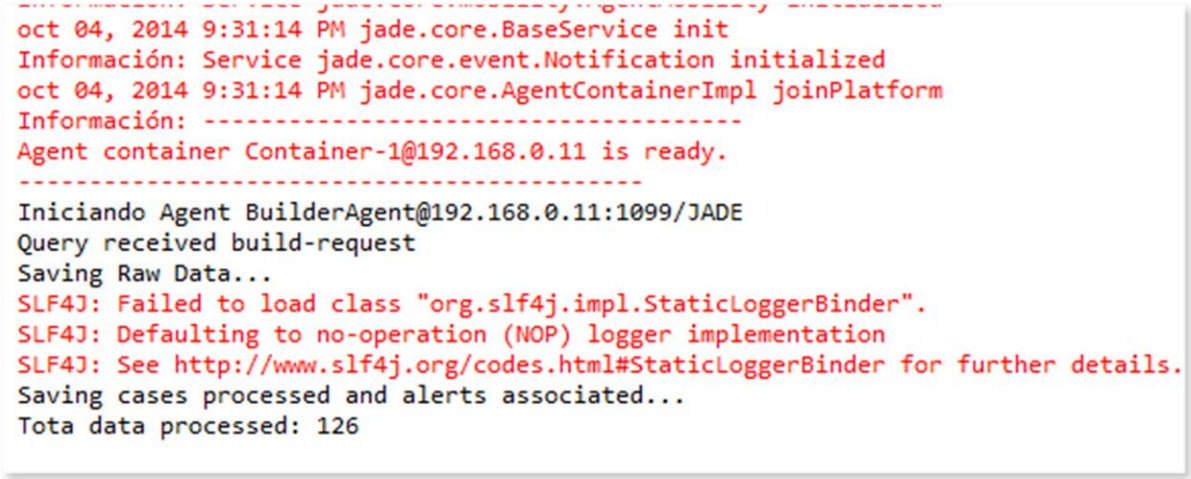

Figura 63. Consola de Ejecución Agente Builder

Cuando se consulta la base de datos el proceso ha sido exitoso. En la figura 64, se visualiza la tabla de eventos con información ya procesada.

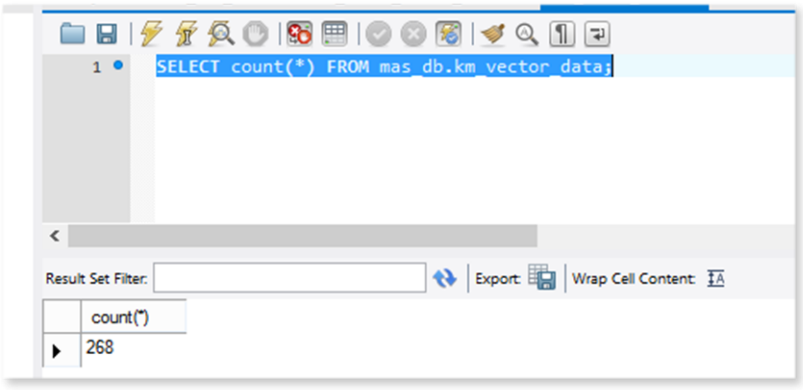

Figura 64. Consulta Base de Conocimiento - Posterior a Agentes

En la figura 65, se presenta la tabla de firmas asociadas a los eventos.

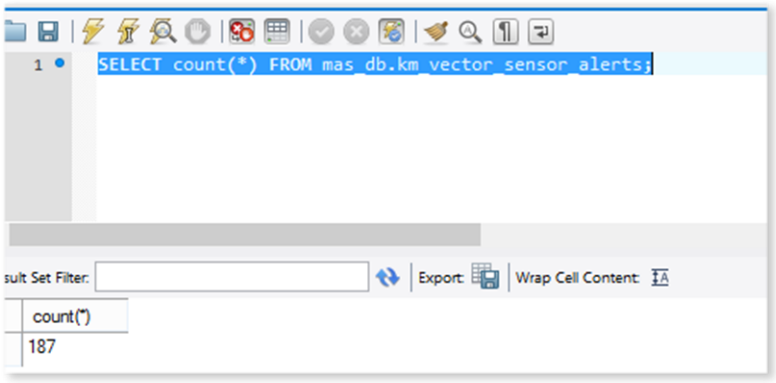

Figura 65. Consulta Base de Conocimiento - Posterior a Agentes 2

3. Ejecución CBR: Se realiza la configuración de la búsqueda. En la figura 66, presenta la configuración base para lanzar el CBR.

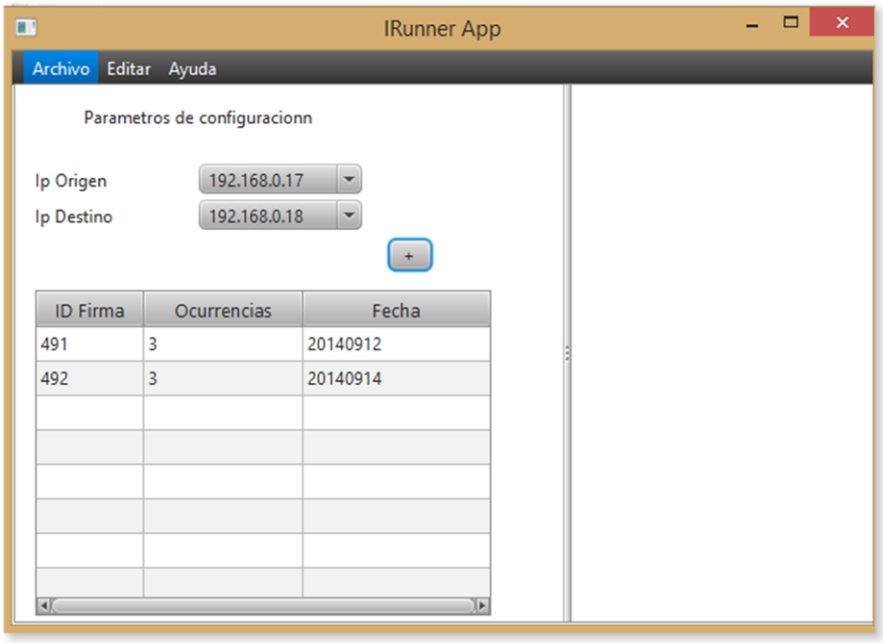

Figura 66. Configuración Consulta - Prueba Integral

Al lanzarse se revisa el log en modo *debug* para revisar el comportamiento. En la figura 67, se observan las consultas y operaciones sobre la base de datos, durante la ejecución del CBR.

```
Hibernate: insert into mas_db.km_vector_data (vector_data_ip_src, vector_data_ip_dst) values (?, ?)
Hibernate: update mas_db.km_vector_sensor_alerts set vector_sensor_alerts_vector_id_real=?, vector_sensor_alerts_sig_id=?, vector_se
Hibernate: update mas_db.km_vector_sensor_alerts set vector_sensor_alerts_vector_id_real=?, vector_sensor_alerts_sig_id=?, vector_se
Hibernate: update mas_db.km_vector_sensor_alerts set vector_sensor_alerts_vector_id_real=?, vector_sensor_alerts_sig_id=?, vector_se
Hibernate: insert into mas_db.km_vector_data (vector_data_ip_src, vector_data_ip_dst) values (?, ?)
Hibernate: insert into mas_db.km_vector_data (vector_data_ip_src, vector_data_ip_dst) values (?, ?)
Hibernate: insert into mas_db.km_vector_data (vector_data_ip_src, vector_data_ip_dst) values (?, ?)
Hibernate: insert into mas_db.km_vector_data (vector_data_ip_src, vector_data_ip_dst) values (?, ?)
22:02:07,916  INFO DataBaseConnector:281 - 5 cases stored into the database.
```

Figura 67. Log de actualización, base de conocimiento

En la figura 68, se aprecia el resultado de la ejecución del sistema multi-agente y de la consulta lanzada por el CBR.

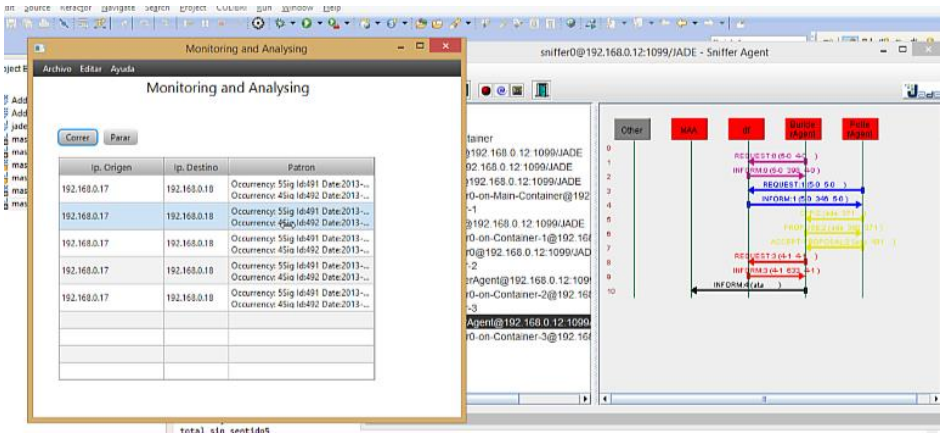

Figura 68. Resultado ejecución sistema multi-agente

En la figura 69, se observa el total de registros después de la ejecución.

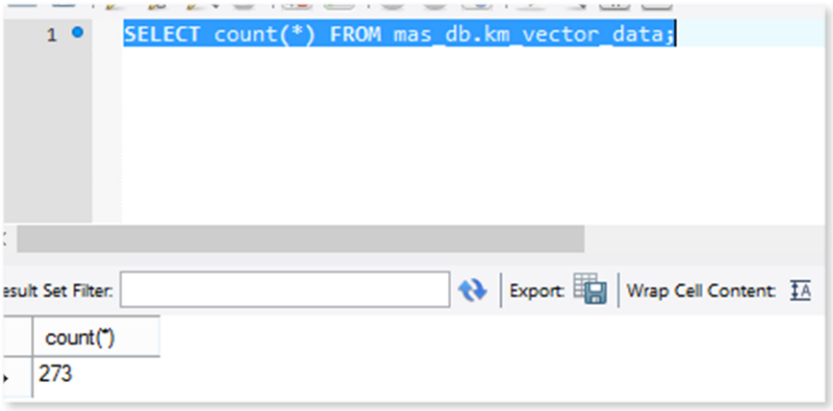

Figura 69. Base de Conocimiento con Datos - Prueba Integral

En la figura 70, se muestra los nuevos casos adicionados a la nueva base del conocimiento.

273	192.168.0.17	192.168.0.18
272	192.168.0.17	192.168.0.18
271	192.168.0.17	192.168.0.18
270	192.168.0.17	192.168.0.18
269	192.168.0.17	192.168.0.18

Figura 70. Base de Conocimiento con Datos - Prueba Integral 2

3.2.3 Fase *Postmortem*.

Se revisa el proceso de desarrollo y las actividades asociadas. Con base a lo anterior se identifica y garantiza la realidad en los formatos de planeación, registro de tiempos y detección de defectos.

Revisando el comportamiento durante los diferentes programas realizados se obtiene el siguiente comportamiento:

Tabla 15 *Tiempos (días) de Fases en PSP para la Plataforma*

Programa	Planeación	Diseño	Codificación	Compilación	Pruebas
P1	42	31	26	1	25
P2	22	73	29	1	10
P3	31	14	29	1	25
P4	1	1	15	1	3

En la figura 71, se presenta el resultado de tiempos en cada una de las fases de PSP.

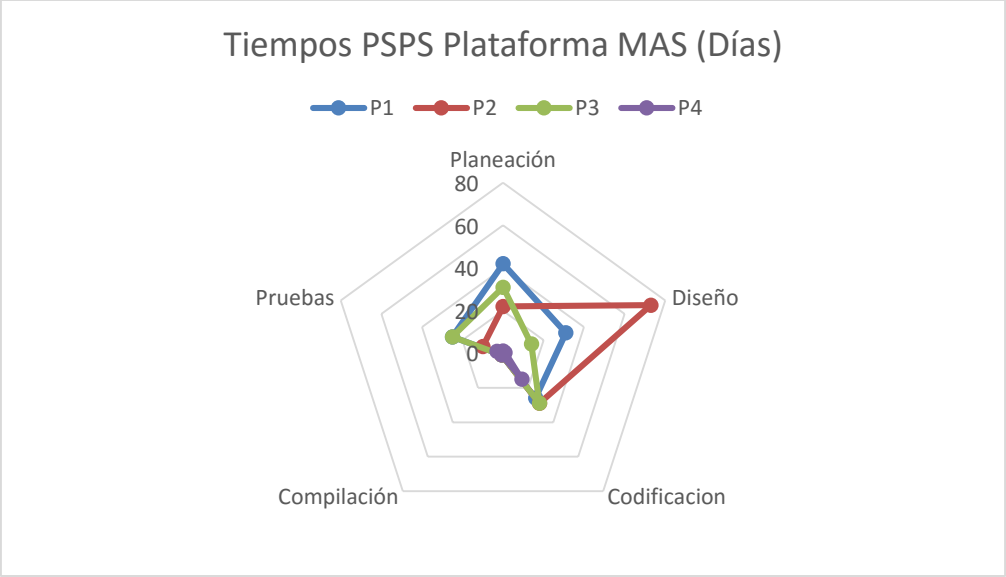

Figura 71. Tiempos PSPS Plataforma MAS (Días)

De lo anterior se puede concluir lo siguiente:

- Los tiempos se redujeron notablemente entre cada programa realizado, esto se debió a que la experiencia el conocimiento en las plataformas utilizadas y la solución a errores comunes creció.
- El programa 2 (Agente Builder) se vio afectado en el diseño por interrupciones ajenas al desarrollo del proyecto, por esto, se ve una desviación sobre la media del diseño.
- Los tiempos más altos se pueden visualizar durante la planeación y la codificación, puntos críticos que definieron en gran medida el diseño y la implementación.

3.3 Análisis de Resultados

Durante el proyecto se ha podido visualizar gran parte de los resultados obtenidos con respecto a los objetivos planteados, esto, tanto en la creación e implementación del sistema multi-agente deliberativo como en su ejecución y clasificación de patrones. Para las pruebas se configuraron

dos redes diferentes, cada una con un *HoneyWall*, una máquina víctima y una máquina atacante, esto acorde a la infraestructura planteada durante el desarrollo.

La ejecución bajo las condiciones observadas dio como resultado:

Tabla 16 *Eventos detectado para K = 10*

Escenario	MAS+CBR
Total Eventos detectados	500
Total detección patrones	80

Como se puede observar, de un total de 500 eventos detectados en ambos entorno, se detectaron 80 patrones con dos ejecuciones acorde a la siguiente configuración y peso:

Tabla 17 *Configuración Base de CBR*

Configuración	ip_src	dst_ip	alerts	Función
Configuración 1	1	1	1	Cadena|Set
Configuración 2	1	1	0,5	Cadena|Set
Configuración 3	0	1	1	Cadena|Set
Configuración 4	1	0	0,5	Cadena|Set

Tabla 18 *Configuración 1*

Id	Descripción	Eventos	Ip origen	Ip destino
365	ICMP	10	192.168.113.132	192.168.0.18
366	ICMP	10	192.168.0.17	192.168.0.18
469	ICMP - Nmap	10	192.168.0.17	192.168.0.18

Tabla 19 *Configuración 2*

Id	Descripción	Eventos	Ip origen	Ip destino
718	Telnet	10	192.168.0.17	192.168.0.18
492	Telnet	10	192.168.0.17	192.168.0.18
716	Telnet	1	192.168.0.17	192.168.0.18

Tabla 20 *Configuración 3*

Id	Descripción	Eventos	Ip origen	Ip destino
628	Port Scan	2	192.168.0.17	192.168.0.18
629	Port Scan - OS	1	192.168.0.17	192.168.0.18
16708	MySql Exploit	1	192.168.0.17	192.168.0.18

Tabla 21 *Configuración 4*

Id	Descripción	Eventos	Ip origen	Ip destino
628	Port Scan	2	192.168.0.17	192.168.0.18
629	Port Scan - OS	1	192.168.0.17	192.168.0.18
8708	Wordpress	1	192.168.0.17	192.168.0.18

Como se observa en la especificación de la configuración, se varían los pesos de cada ítem configurado para validar la obtención del resultado. Es importante tener en cuenta que las alertas se representan por un conjunto de eventos definidos cuya función de similitud define cuan similar son a la base capturada.

La siguiente tabla muestra el comportamiento de los patrones detectados

Tabla 22 *Detección Correcta vs Detección Fallida*

	Configuración 1	Configuración 2	Configuración 3	Configuración 4
Vectores Acertados	90%	20%	90%	30%
Vectores Incorrectos	10%	80%	10%	70%

En la figura 72, se presenta el resultado de detección correcta e incorrecta de nuevos casos.

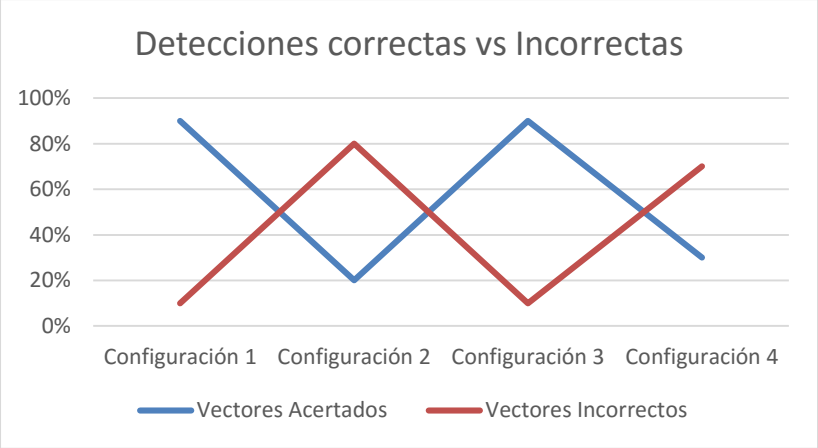

Figura 72. Comportamiento de la Detección

El comportamiento anterior muestra una alta asertividad en las configuraciones de clasificación en donde los atributos son tipos básicos como cadenas de texto; comportamiento diferente en el momento que se utiliza la función de configuración asociada al listado de alertas en donde se utilizó la intersección de los objetos sin especificar el detalle de la comparación.

4. CONCLUSIONES

Al realizar el proceso de implementación del sistema multi-agente con base un modelo estándar de base de datos sobre el cual se puedan ejecutar procesos de extracción y procesamiento de datos, se pudieron identificar diferentes tipos de vectores de ataque e información anómala, con una base de conocimiento previamente cargada a partir de eventos de diferentes entornos, mostrando que a partir de una configuración previa se pudo lograr una deliberación acertada de nuevos patrones de ataques.

El sistema multi-agente deliberativo se basó en la implementación de un CBR, el cual apoyo de manera correcta la parte de razonamiento del proyecto y de los agentes, sin embargo, las funciones de comparación que se desarrollaron han mostrado comportamiento no acertados en el momento de realizar la ejecución del algoritmo de clasificación, por lo cual, la mejora de estas es un punto crítico que se debería de mejorar en la implementación, aumentando la capacidad de deliberación correcta tanto de vectores de ataque como de información anómala.

La utilización de agentes inteligentes bajo JADE mostro un desempeño relativamente bueno y acorde al estándar FIPA-ACL de comunicación, ayudando el trabajo colaborativo de los agentes para lograr la extracción, envío, procesamiento y deliberación sobre la información que se obtuvo de las diferentes fuentes, sin embargo, el proceso de envió de los eventos identificados en los entornos *HoneyNet* es un factor por mejorar, teniendo en cuenta que dentro del mensaje enviado acorde al estándar FIPA-ACL, fue necesario el uso de un texto formateado en el estándar JSON para su envió e interpretación por los demás agentes, haciendo en ocasiones, lento el envío y recepción de la información.

Aun cuando la implementación de un algoritmo de aprendizaje supervisado como es CBR con un algoritmo de clasificación KNN para obtener un resultado de deliberación sobre información recolectada presento un comportamiento aceptable; dependía obviamente de una buena base de conocimiento, que para este caso, no estaba afinada en un porcentaje superior al 80% para obviar falsos positivos, generando en algunos casos, falsas deliberaciones sobre nuevos vectores de ataque y datos anómalos.

5. CUMPLIMIENTO DE OBJETIVOS

- Modelar un sistema deliberativo basado en agentes, basado en sensores *HoneyNet* que cumpla con los estándares de esta iniciativa.

  La iniciativa bajo el proyecto *HoneyNet* brindo los lineamientos base para poder realizar y moldear un entorno vulnerable que garantiza la captura, almacenamiento, análisis y visualización de los eventos detectados. Para lograr modelar el entorno base para el sistema multi-agente, se utilizó *security onion* con *Snort* como IDS principal, logrando simular ataques reales en un entorno controlado trampa señuelo y buenos resultados con un coste bajo a nivel de recursos de máquina, esto puede ser visualizado dentro de la explicación del entorno vulnerable, especificando las máquinas y características que se utilizaron. Una vez se generó el entorno, la revisión del esquema de base de datos asociado a los eventos y su asociación tanto al modelo de datos puro como a la generación de casos base para la implementación del CBR fue relativamente sencillo, en la medida que los procesos de transformación fueron definidos de manera correcta.

  Teniendo la configuración base el modelamiento del comportamiento deseado se definió bajo el estándar FIPA-ACL, estándar que definió la manera de construir los agentes

y cómo interactúan entre sí, con el objetivo social de obtener y procesar los eventos detectados dentro de la *HoneyNet*. Para materializar este modelamiento se utilizó JADE, componente de capa media que cumple con los lineamientos asociados y permite el desarrollo y ejecución de agentes.

- Analizar, diseñar, desarrollar e implementar un sistema multi-agente deliberativo y distribuido basado en JADE, que permita la obtención y análisis de datos de *HoneyNets*.

Durante el desarrollo del proyecto se puede visualizar, en sus fases de análisis, diseño, desarrollo y pruebas la construcción de los agentes encargados de la consulta de los eventos asociados a los ataques realizados dentro del entorno vulnerable, al igual que su procesamiento y envío al agente constructor para que alimenta la base de conocimiento. Lo anterior y el uso de agentes para el desarrollo del proyecto garantizó cualidades vitales como el uso en diferentes locaciones geográficas mediante la generación de un punto centralizado o contenedor en donde se deben de registrar los agentes, modelar una sociedad basada en oferta y demanda de servicios, autonomía en la respuesta asociada al estado actual del contenedor y el rendimiento que puede ofrecer.

La plataforma JADE ofreció las características necesarias para desarrollar una solución basada en Agentes, dividiendo las responsabilidades acorde al dominio de cada uno, con miras a cumplir el objetivo de suplir con los datos necesarios para generar la base de conocimiento, mediante el procesamiento de los eventos generados y el almacenamiento en la base de datos local. Adicional a lo anterior, ofrece ventajas en el descubrimiento y registro de servicios, movilidad e interoperabilidad en diferentes plataformas y dispositivos, requerimientos en infraestructura y librerías, utilidades incluidas para ejecutar algoritmos de clasificación y búsqueda. Debido a lo anterior fue muy sencillo, una vez

desarrollados los agentes, ejecutarlos en diferentes sistemas operativos, siempre y cuando tuvieran el JRE de Java instalado, sin requisitos elevados o específicos a nivel de infraestructura.

Este componente de capa media al estar desarrollado en Java permitió que su integración con el esquema de base de datos asociado al IDS fuera muy simple, al igual que con otros librerías de código abierto para realizar las transformaciones necesarias para el almacenamiento de los datos. Durante las transformaciones se identifica que el uso de la especificación JSON para el envío de datos no siempre es la más eficiente, por lo cual se podría evaluar el uso de otra estrategia.

- Validar el comportamiento deliberativo de los diferentes vectores de ataque frente a patrones y vectores ya conocidos.

  Durante el proyecto se realizó la ejecución de varios escenarios, todos partiendo de una configuración base de clasificación y búsqueda para categorizar los patrones que se encuentran durante la ejecución del sistema deliberativo, esto, logro obtener un comportamiento específico en cuanto a casos para ser reutilizados en donde el factor de comparación de las alertas determinó en gran medida la asertividad, dando como resultado un comportamiento en picos. Para lograr y validar el comportamiento deliberativo se utilizó el algoritmo KNN que se encuentra implementado dentro de la jColibri, librería que permitió implementar el comportamiento deliberativo; sin embargo, este se basa en funciones de comparación de similitud, una de las cuales se implementó para esta plataforma y que valdría la pena mejorar para aumentar el porcentaje de asertividad.

6. RECOMENDACIONES

- El proyecto como tal puede extenderse mediante el cambio del algoritmo de clasificación KNN supervisado por uno no supervisado de aprendizaje como es el *Clustering*[38], de esta manera se podría lograr un aprendizaje y clasificación autónoma, con una mejora iterativa incremental sin depender realmente de una base de conocimiento.
- Aun cuando se logra identificar nuevos patrones e información anómala, podría llegar a implementarse un motor de correlación de eventos a partir de nuevos patrones y existentes, mostrando desde un panorama más global que conjunto de eventos o patrones podrían generar un ataque, o empezar a entender el comportamiento más afondo de los atacantes.
- Para la implementación, la base de conocimiento se hace relativamente importante para la correcta deliberación e identificación de nuevos patrones, por lo cual, se recomienda que la fuente de información este correctamente afinada y el proceso de extracción por el agente correspondiente correctamente validado.
- El uso de agentes inteligentes bajo una plataforma de investigación y educativa ofrece resultados aceptables, sin embargo, el extender o implementar una plataforma con características orientadas a cumplir atributos de calidad de la industria actual, podría mejorar no solo el rendimiento, si no la implementación y la introducción real al mercado de este tipo de tecnologías.
- El estándar de agentes actual cumple en gran medida con las características de una arquitectura orientada a servicios, ya que conceptualmente los agentes fueron creados para ser autónomos, transparencia en su locación, granulares, reusables y descubribles; por lo

[38] Estrategia de clasificación, basada en agrupamiento.

cual, trabajar en sustentar o como suplir las demás características como interoperabilidad que sean sin estado, puede aportar a la industria un gran avance y lograr una implementación nativa y totalmente acorde a los patrones que acompañan una arquitectura SOA[39] (*Service Oriented Architecture*).

- Es importante re-definir las funciones de Similitud que soportan la clasificación de los eventos para que la deliberación sea mucho más acertada, esto se puedo lograr mediante una mejora significativa en el algoritmo de comparación de los patrones de ataques.
- Los agentes basan su funcionamiento en sus características de autonomía, proactividad, reactividad, entre otras, por lo cual, es importante entender y diseñar la arquitectura de agentes basadas en esto, sacando provecho de sus características sin entorpecer otros atributos de calidad.
- Diferenciar y entender la ventaja que tiene la implementación de un sistema multi-agente desde su protocolo de comunicación y características sociales, garantiza desde un inicio el foco que se le debe de dar y como sacar ventaja de esto.
- Actualmente existen esfuerzos para entregar listo entornos que apoyan diferentes dominios informativos, hacer un buen análisis de herramientas existentes y cuál es la mejor opción garantiza una disminución considerable de tiempos.

[39] Arquitectura de referencia para la implementación de componentes reutilizables, entre otras características.

BIBLIOGRAFÍA

Aamodt, A., & Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Al Communications*, 7(1), 39-59.

Alonso A., Porras, S., Ezpeleta, E., Vergara, E., Arenaza, I., Uribeetxeberria, R., … Corchado, E. (2010). Understanding Honeypot Data by an Unsupervised Neural Visualization. En Springer, *Computational Intelligence in Security for Information System 2010* (pp. 151-160). Springer Berlin Heidelberg.

Becker, R.A, Eick, S.G., & Wilks, A.R., (1995). Visualizing Network Data. *Transaction on Visualization and Computer Graphic*, *IEEE Transaction on*, *1*(1), 16-28. doi: 10.1109/2945.468391

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing Multi- agent systems with JADE.*, West Sussex, England: Wiley Series.

Bratman, M. (1987). *Intentions, Plans, and Practical Reason.* Cambridge, Massachusetts: Harvard University Press

Buttle, F. (2009). *Introduction.* Customer Relationship Management, (Concepts and Technologies). Burlington, MA: Elsevier Ltda

Camurri, A. & Coglio, A. (1998). *An Architecture for Emotional Agents*. IEEE Multimedia, 5(4), 24-33

Carneige Mellon University. (2006). *Personal Software Process for Engineers, Using PSP0*. Recuperado de: http://www.sei.cmu.edu

Castillo, A. (2004). *Modelos y Plataformas de Agentes Software Móviles e Inteligentes para Gestión del Conocimiento en el Contexto de las Tecnologías de la Información* (Tesis Doctoral). Universidad Pontificia de Salamanca. Madrid.

Döring, C. (2005). *Improving network security with Honeypots, Honeypots Project* (Tesis de

Maestría), Universidad Wisconsin- Platteville. Darmstadt.

Ferber, J. (1999). *Multi-Agent Systems. An introduction to distributed artificial intelligence*. Addison-Wessley.

Fipa, (2003). *"The Foundation for Intelligent Physical Agents"*. FIPA (The Foundation for Intelligent Physical Agents. Recuperado de http://www.fipa.org

Herrero, A., & Corchado, E. (2009). Multiagent System for Network Intrusion Detection: A Review. In *Computational Intelligence in Security for Information System*. 143-154. Springer Berlin Heidelberg.

Highsec. (2013, 11 de Julio). *Conociendo Metasploit – Parte I – Exploit Basico* [web log post]. Recuperado de http://highsec.es/2013/07/conociendo-metasploit-parte-i-exploit-basico/

HoneyNet Project. (2006). *Know your enemy: HoneyNets*. Honeynet Project. Recuperado de http://old.honeynet.org/papers/honeynet/

Humphrey, W. S. (2000). *The Personal Software Process^{SM} (PSPSM*). Recuperado de http://www.sei.cmu.edu/reports/00tr022.pdf

Isaza, G. (2010). *Modelo de Sistema multiagente de detección y prevención de intrusiones basado en inteligencia Computacional híbrida y representaciones ontosemanticas* (Tesis Doctoral). Universidad Pontificia de Salamanca. Madrid.

Isaza, G. A, Castillo, A. G., & Segura, A.A. (2008). Modelo de Detección de Intrusos basado en Sistemas Multi-agente, inteligencia Computacional y representaciones Ontológicas. *X RECSI - X Reunión Española sobre Criptología y Seguridad de la Información*. (1), 271-282.

Khosravifar, B., Gomrokchi, M., & Bentahar, J. (2009). A Multi-Agent-Based Approach to Improve Intrusion Detection Systems False Alarm Ratio by Using Honeypot. *Advanced Information Networking and Application Workshops, 2009 Waina´09. International Conference on*. 97-102. doi: 10.1109/WAINA.2009.103

Kim, I.S., & Kim, M.H. (2012). Agent-Bases Honeynet Framework for Protecting Server in

Campus Networks. *Information Security, IET*, 6(3), 202-211. doi:10.1049/iet-ifs.2011.0154

Krmicek, V., Celeda, P., Rehak, M., & Pechoucek, M., (2007) Agent- Based Network Intrusion Detection System. *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, *IEEE Computer Society*, 528- 531. doi: 10.1109/IAT.2007.111

Kumar, S. (1995). *Classification and Detection of Computer Intrusions* (Tesis Doctoral). Universidad de Purdue.

Lips, R. & El-Kadhi, N. (2008) Intelligent Mobile Agent for Intrusion Detection System (IMAIDS). European Institute of Technology. Le Kremlin-France. Obtenido en: http://citeseerx.ist.psu.edu/viewdoc/summary

Pan, Z.S, Chen, S.C., Hu, G.B, & Zhang, D.Q. (2003). Hybrid neural network and C4.5 for misuse detection. *International Conference on Machine Learning and Cybernetics*, (4), 2463 - 2467. doi: 10.1109/ICMLC.2003.1259925

Pinzón, C., Herrero, A., De Paz, J. F., Corchado, E, & Bajo, J. (2010). CBRid4SQL: A CBR Intrusion Detector for SQL Injection Attacks. En E. Corchado, M. Granada & A. Manhaes. (Ed), *Hybrid Artificial Intelligence Systems* (510 - 519). Springer.

Pomeroy-Huff, M., Cannon, R., Chick, T.A, Mullaney, J., Nichols, W. (2009). *The Personal Software Process^{SM} (PSP^{SM}) Body of Knowledge, Version 2.0*. Recuperado de http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8907

Poslad, S. (2007). Specifying Protocols for Multi-Agent Systems Interaction. *Association for Computing Machinery*. Recuperado de http://www.fipa.org/subgroups/ROFS-SG-docs/2007-TAAS-specifying-MAS.pdf

Rammidi, G. (2010). *Survey on Current Honeynet research*. Obtenido en: http://honeynetproject.ca/files/survey.pdf

Recio-García, JA., Díaz-Agudo, B., & González- Calero, P. (2008). *jCOLIBRI2 Tutorial. Group for Artificial Intelligence Applications*, Universidad Complutense de Madrid. Recuperado de http://gaia.fdi.ucm.es/files/people/juanan/jcolibri/downloads/tutorial.pdf

Sanjuán, O. (2006). *Métodos Evolutivos para la Construcción de Sistemas de Agentes Adaptativos en Entornos Cambiantes y Heterogéneos* (Tesis Doctoral). Universidad Pontificia de Salamanca**.** Madrid.

Tecuci, G. (1998). *Building Intelligent Agents. An Apprenticeship, multistrategy learning theory, methodology, tool and case studies.* Academic Press.

Wang, H. & Chen, Q. (2010). Design of Cooperative Deployment in Distributed Honeynet System. *Computer Supported Cooperative Word in Design (CSCWD), 2010 14th International Conference on*, 711-716. doi:10.1109/CSCWD.2010.5471884

Xin, J., Dickerson, J. E. & Dickerson, J. A. (2003). Fuzzy feature extraction and visualization for intrusion detection. *The 12th IEEE International Conference on Fuzzy Systems*, 2, 1249 – 1254. doi: 10.1109/FUZZ.2003.1206610

Yin, J., Zhang, G., & Chen, Y-Q. (2003, Nov. 2-5). Intrusion discovery with data mining on Honeynet, *Machine Learning and Cybernetics, 2003 International Conference on* (1), 41-45. doi: 10.1109/ICMLC.2003.1264439

APÉNDICE

Apéndice A. Presupuesto

A1. Resumen por Rubros.

Tabla 23 *Resumen de Rubros*.

Rubros	Total
EQUIPOS	$ 1.350.000
PERSONAL	$ 1.200.000
MATERIALES	$ 150.000
SERVICIOS TÉCNICOS	$ 348.000
BIBLIOGRAFIA	$ 225.000
TOTAL	$ 3.275.000

A2. Presupuesto Detallado.

- Equipos

Tabla 24 *Presupuesto de Equipos.*

Descripción	Justificación	Cantidad	Valor unitario	Total
Equipo portátil	Herramienta de trabajo para el profesional encargado de realizar el proyecto de grado.	1	$ 1.350.000	$ 1.000.000
Totales				$ 1.350.000

- Personal

Tabla 25 *Presupuesto de Personal.*

Descripción	Justificación	Cantidad /Personas	Valor unitario	Dedicación	Total
			Horas	Horas	
Profesional	Desarrollo de los objetivos planteados en el proyecto	1	$ 0	250	$ 0
Asesor	Asesoría Temática y Técnica durante el proceso de investigación	1	$ 6.000	200	$ 1.200.000
Totales					$ 1.200.000

- Materiales

Tabla 26 *Presupuesto de Materiales.*

Descripción	Justificación	Cantidad	Valor unitario	Total
Papelería, fotocopias, tinta, USB, DVD, ganchos, Lapiceros	Requerido para el proyecto	1	$ 150.000	$ 150.000
Totales			$ 150.000	$ 150.000

- Servicios Técnicos

Tabla 27 *Presupuesto de Servicios Técnicos.*

Descripción	Justificación	N. Meses	Valor	Total

			Mensual	
Conexión internet	conectividad de datos	12	$ 58.000	$ 696.000
Totales			$ 58.000	$ 696.000

- Bibliografía

Tabla 28 *Presupuesto de Bibliografía.*

Inteligencia Competitiva	**Justificación**	**Cantidad**	**Valor**	**Total**
Libros y Artículos	Material de Consulta en inteligencia artificial y desarrollo de agentes.	3	$ 75.000	$ 225.000
Totales			$ 75.000	$ 225.000

Apéndice B. SourceCode - mas_data_core

```
package co.emgiraldo.xml.service;


import com.google.gson.Gson;
import com.google.gson.JsonSyntaxException;

public class MarshallingService<T> {

    private Gson gson;

    public MarshallingService(){
        gson = new Gson();
    }

    public String marshalData(T clas){
        return getGson().toJson(clas);
    }

    @SuppressWarnings("unchecked")
    public T unmarshalData(String json, T clas){
        try{
            return (T) getGson().fromJson(json, clas.getClass());
        }catch(JsonSyntaxException ex){
            System.out.println("Error parsing "+json+"\n"+ex.getMessage());
        }
        return null;
    }

    public Gson getGson() {
        return gson;
    }

    public void setGson(Gson gson) {
        this.gson = gson;
    }
}
```

```
package co.emgiraldo.data.dto;

import java.sql.Date;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="alert")
public class SnortAlertDto {

    private int sid;
    private int cid;
    private String sigName;
    private int signatureId;
    private Date timestamp;
    private int sigPriority;
    private int sigRev;
    private int eventsCount;
    private String ipSrc;
    private String ipDst;
    private int ipVer;
    private int ipHlen;
    private int ipTos;
    private int ipLen;
    private int ipId;
    private int ipFlags;
    private int ipOff;
    private int ipTtl;
    private int ipProto;
    private int ipCsum;
    private String dataPayload;

    public int getSid() {
        return sid;
    }

    @XmlElement
    public void setSid(int sid) {
        this.sid = sid;
    }
    public int getCid() {
        return cid;
    }
    @XmlElement
    public void setCid(int cid) {
        this.cid = cid;
    }
    public String getSigName() {
        return sigName;
    }
    @XmlElement
    public void setSigName(String sigName) {
        this.sigName = sigName;
    }
    public int getSignatureId() {
        return signatureId;
    }
```

```
public Date getTimestamp() {
        return timestamp;
    }
    @XmlElement
    public void setTimestamp(Date timestamp) {
        this.timestamp = timestamp;
    }
    public int getSigPriority() {
        return sigPriority;
    }
    @XmlElement
    public void setSigPriority(int sigPriority) {
        this.sigPriority = sigPriority;
    }
    public int getSigRev() {
        return sigRev;
    }
    @XmlElement
    public void setSigRev(int sigRev) {
        this.sigRev = sigRev;
    }
    public int getEventsCount() {
        return eventsCount;
    }
    @XmlElement
    public void setEventsCount(int eventsCount) {
        this.eventsCount = eventsCount;
    }
    public String getIpSrc() {
        return ipSrc;
    }
    @XmlElement
    public void setIpSrc(String ipSrc) {
        this.ipSrc = ipSrc;
    }
    public String getIpDst() {
        return ipDst;
    }
    @XmlElement
    public void setIpDst(String ipDst) {
        this.ipDst = ipDst;
    }
    public int getIpVer() {
        return ipVer;
    }
```

```
package co.emgiraldo.poller.sql.connection.mgmnt.boundary;

import java.sql.ResultSet;
import java.util.List;

import co.emgiraldo.data.dto.SnortAlertDto;
import
co.emgiraldo.poller.sql.connection.mgmnt.constants.SnortSqlQueryConstants;
import co.emgiraldo.poller.sql.connection.mgmnt.service.ConnectionService;
import co.emgiraldo.poller.sql.connection.mgmnt.service.QueryService;
import co.emgiraldo.poller.sql.connection.mgmnt.service.TransformationService;

public class DbService {

    private ConnectionService connService;
    private QueryService queryService;
    private TransformationService transformationService;

    public DbService(){
        connService = new ConnectionService();
        queryService = new QueryService();
        transformationService = new TransformationService();
    }

    /**
     * Method for querying alerts captured by HoneyNet Sensors (Snort).
     *
     * @param full {@link Boolean} Indicates if it is a full recover or a
incremental recovery
     *
     * @return {@link List} The list of Snort Alerts
     */
    public List<SnortAlertDto> querySnortRawData(boolean full){
        ResultSet rs = null;
        if(full){
            rs = getQueryService().executeSqlQueryWithReturnedValues(
                    SnortSqlQueryConstants.QUERY_SNORT_RAW_DATA,
getConnService().getConnectionMySql());
        }else{
            rs = getQueryService().executeSqlQueryWithReturnedValues(
                    SnortSqlQueryConstants.QUERY_SNORT_RAW_DATA_ONE_MINUTE,
getConnService().getConnectionMySql());
        }
        return getTransformationService().convertSnortQueryRsToVo(rs);
    }

    public ConnectionService getConnService() {
        return connService;
    }

    public void setConnService(ConnectionService connService) {
        this.connService = connService;
    }

    public QueryService getQueryService() {
        return queryService;
    }
```

```
package co.emgiraldo.poller.sql.connection.mgmnt.service;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;



public class ConnectionService {

    private static final String DRIVER_NAME = "com.mysql.jdbc.Driver";
    private static final String DB_SERVER = "192.168.0.10";
    private static final String PORT_NOMBER = "3306";

    private Connection connectionMySql;

    private String getDriverUrl(){
        return "jdbc:mysql://"+DB_SERVER+":"+PORT_NOMBER+"/snorby";
    }

    public ConnectionService(){
        setupConnection();
    }

    private void setupConnection() {
        try {
            if(connectionMySql == null || connectionMySql.isClosed()){
                Class.forName(DRIVER_NAME);
                connectionMySql =
DriverManager.getConnection(getDriverUrl(),"agentmas","agent");
            }
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("Error:
ConnectionService_setupConnection()"+e.getCause().getMessage());
        }
    }

    public void closeConnection(){
        try {
            if(!getConnectionMySql().isClosed()){
                getConnectionMySql().close();
            }
        } catch (SQLException e) {
            System.out.println("Error al cerrar la conexion: "+e.getMessage());
        }
    }

    public Connection getConnectionMySql() {
        setupConnection();
        return connectionMySql;
    }

    public void setConnectionMySql(Connection connectionMySql) {
        this.connectionMySql = connectionMySql;
    }
}
```

```
import java.util.List;

import co.emgiraldo.data.dto.SnortAlertDto;
import co.emgiraldo.poller.sql.connection.mgmnt.boundary.DbService;
import co.emgiraldo.xml.constants.MarshallingConstants;
import co.emgiraldo.xml.service.MarshallingService;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class PollerAgent extends Agent{


    private static final long serialVersionUID = -1514468139134052638L;
    private static final String REQUEST_BUILD = "build-request";

    private DbService dbService;
    private MarshallingService<SnortAlertDto> marshallingService;
    private AID[] builderAgents;

    protected void setup(){
        System.out.println("Iniciando agente
Recolector..."+getAID().getName());
        dbService = new DbService();
        marshallingService = new MarshallingService<>();

        addBehaviour(new PollerBehaviour(this));
    }
```

```
private class PollerBehaviour extends TickerBehaviour{

        private static final long serialVersionUID = -5367974842559047621L;

        public PollerBehaviour(Agent a){
            super(a, 60000);
        }

        @Override
        protected void onTick() {
            System.out.println("Starting Building Process Request...");


            DFAgentDescription template = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType("builder-agent");
            template.addServices(sd);

            try{
                DFAgentDescription[] result = DFService.search(myAgent,
template);
                System.out.println("The following Builder Agents were
Found");
                builderAgents = new AID[result.length];
                for(int i = 0; i < result.length; i++){
                    builderAgents[i] = result[i].getName();
                    System.out.println(builderAgents[i].getName());
                }

            }catch(FIPAException ex){
                System.out.println(ex.getMessage());
            }

            myAgent.addBehaviour(new RequestBuilderPerformer());
        }

    }
```

```
private class RequestBuilderPerformer extends Behaviour{

        private static final long serialVersionUID = 4470692444025198994L;

        private MessageTemplate mt;
        private AID bestBuilder;
        private int bestPerformance;
        private int totalCount = 0;

        private int step = 0;

        @Override
        public void action() {
            switch (step) {
            case 0:
                ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
                for(int i = 0; i < builderAgents.length; i++){
                    cfp.addReceiver(builderAgents[i]);
                }
                cfp.setContent(REQUEST_BUILD);
                cfp.setConversationId("builder-trade");
                cfp.setReplyWith("cfp"+System.currentTimeMillis());
                myAgent.send(cfp);

                mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("builder-trade"),

MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                step = 1;
                break;

            case 1:
                ACLMessage reply = myAgent.receive();
                if(reply != null){
                    if(reply.getPerformative() == ACLMessage.PROPOSE){
                        int averagePerformance =
Integer.valueOf(reply.getContent());
                        if(averagePerformance < 3){
                            if(bestBuilder == null || averagePerformance <
bestPerformance){
                                bestPerformance = averagePerformance;
                                bestBuilder = reply.getSender();
                            }
                        }
                    }
                    totalCount++;
                    if(totalCount >= builderAgents.length){
                        step = 2;
                    }
                }else{
                    block();
                }
                break;

            case 2:
                ACLMessage order = new
```

```
                order.addReceiver(bestBuilder);
                order.setContent(queryInfo());
                order.setConversationId("builder-trade");
                order.setReplyWith("order"+System.currentTimeMillis());
                myAgent.send(order);
                mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("builder-trade"),

MessageTemplate.MatchInReplyTo(order.getReplyWith()));
                step = 3;
                break;
            case 3:
                reply = myAgent.receive(mt);
                if(reply != null){
                    if (reply.getPerformative() == ACLMessage.INFORM) {
                        System.out.println("Building process was performed
by "+reply.getSender().getName());
                        myAgent.doDelete();
                    }
                    step = 4;
                }else{
                    block();
                }
                break;
            default:
                break;
            }

        }
```

Apéndice C. SourceCode - mas_data_builder

```
package co.emgiraldo.builder.datamgmnt.processservice;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import co.emgiraldo.builder.entity.KmRawData;
import co.emgiraldo.builder.entity.KmVectorData;
import co.emgiraldo.builder.entity.KmVectorSensorAlerts;

public class DataProcessService {

    private List<KmVectorData> vectorDataList;

    public DataProcessService(){
        vectorDataList = new ArrayList<>();
    }

    /**
     * Method for processing a list of {@link KmRawData}, the result of
this
     * is a set of data based on source ip and destination ip of an
alert, it
     * has also have the associated alerts.
     *
     * @param dataList {@link List} of {@link KmRawData}
     *
     * @return {@link List} of {@link KmVectorData} containing the final
result
     * of the processed data
     */
    public List<KmVectorData> processData(List<KmRawData> dataList){
        for(KmRawData km:dataList){
            validateKmRawDataItem(km);
        }
        return getVectorDataList();
    }

    private void validateKmRawDataItem(KmRawData rawData){
        int pos = itemHasMatchInVectorData(rawData);
        if(pos != -1){
            processItemAlreadyInList(rawData, pos);
        }else{
            processItemNotInList(rawData);
        }
    }
```

```
/**
     * Method for validating if a KmRawData is already in the local
     * list of KmRayData
     *
     * @param item {@link KmRawData}, the element to validate
     * @return {@link Integer}, It returns the position of the item if
     * it's found, otherwise it returns -1.
     */
    private int itemHasMatchInVectorData(KmRawData item){
        for(int i=0; i < getVectorDataList().size(); i++){

if(getVectorDataList().get(i).getVectorDataIpDst().equalsIgnoreCase(item.ge
tIpDst()) &&

getVectorDataList().get(i).getVectorDataIpSrc().equalsIgnoreCase(item.getIp
Src())){
                return i;
            }
        }
        return -1;
    }

    /**
     * Method for processing an item which is found in the local list, it
validates
     * if the alerts associated match the one is passed in, if so, the
occurrence is
     * increased in one, otherwise a new item is created and added to the
list.
     *
     *
     * @param rawData {@link KmRawData}, the item which a new alert is
going to be added or
     * the alert to be increased in occurrences.
     * @param pos {@link Integer}, the position in the list where element
can be get
     */
    private void processItemAlreadyInList(KmRawData rawData, int pos){
        KmVectorData itemInList = getVectorDataList().get(pos);
        Iterator<KmVectorSensorAlerts> it =
itemInList.getKmVectorSensorAlertses().iterator();
        KmVectorSensorAlerts iteratorItem = null;
        boolean found = false;
        while(it.hasNext()){
            iteratorItem = it.next();

if(iteratorItem.getVectorSensorAlertsSigId().compareTo(rawData.getSignature
Id()) == 0){

iteratorItem.setVectorSensorAlertsOcurrency(iteratorItem.getVectorSensorAle
rtsOcurrency()+1);
                found = true;
                break;
            }
        }
```

```
if(!found){
            iteratorItem = new KmVectorSensorAlerts();
            iteratorItem.setVectorSensorAlertsOcurrency(1);

iteratorItem.setVectorSensorAlertsSigId(rawData.getSignatureId());

iteratorItem.setVectorSensorAlertsTimestap(rawData.getTimestamp().toString(
));
            found = false;
        }
        iteratorItem.setKmVectorData(itemInList);

getVectorDataList().get(pos).getKmVectorSensorAlertses().add(iteratorItem);
    }

    /**
     * Method for creating a new element and an alert associated
     *
     * @param rawData {@link KmRawData}, the element for creating a new
{@link KmVectorData}
     */
    private void processItemNotInList(KmRawData rawData){
        KmVectorData vectorData = new KmVectorData();
        KmVectorSensorAlerts sensorAlerts = new KmVectorSensorAlerts();
        vectorData.setVectorDataIpDst(rawData.getIpDst());
        vectorData.setVectorDataIpSrc(rawData.getIpSrc());

        sensorAlerts.setVectorSensorAlertsOcurrency(1);
        sensorAlerts.setVectorSensorAlertsSigId(rawData.getSignatureId());

sensorAlerts.setVectorSensorAlertsTimestap(rawData.getTimestamp().toString(
));
        sensorAlerts.setKmVectorData(vectorData);

        Set<KmVectorSensorAlerts> setTmp = new HashSet<>();
        setTmp.add(sensorAlerts);

        vectorData.setKmVectorSensorAlertses(setTmp);

        getVectorDataList().add(vectorData);
    }

    private List<KmVectorData> getVectorDataList() {
        return vectorDataList;
    }
}
```

```
package co.emgiraldo.builder.dao.crudservice;

import java.util.List;

import org.hibernate.Query;

import co.emgiraldo.builder.db.util.HibernateUtil;

public class GenericDaoService<T> {

    public boolean persistObject(T clazz){
        try{

HibernateUtil.getSessionFactory().getCurrentSession().beginTransaction();

HibernateUtil.getSessionFactory().getCurrentSession().persist(clazz);

HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().comm
it();
            return true;
        }catch(Exception ex){
            System.out.println("Error persisting
"+clazz.getClass().getCanonicalName()+" - " +ex.getMessage()+" - - ");
            ex.printStackTrace();

HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().roll
back();
        }finally{
            HibernateUtil.getSessionFactory().close();
        }
        return false;
    }

    public List<T> findAll(T clazz){
        HibernateUtil.getSessionFactory().openSession();

HibernateUtil.getSessionFactory().getCurrentSession().beginTransaction();
        Query query =
HibernateUtil.getSessionFactory().getCurrentSession().createQuery("from
KmRawData");
        @SuppressWarnings("unchecked")
        List<T> listTmp = (List<T>)query.list();
        return listTmp;
    }
}
```

```
package co.emgiraldo.builder.dao.crudservice;

import co.emgiraldo.builder.entity.KmVectorSensorAlerts;

public class VectorSensorAlerts extends
GenericDaoService<KmVectorSensorAlerts>{

}
```

```
public class BuilderAgent extends Agent{

    private static final long serialVersionUID = 5360305558843904287L;
    private static final String REQUEST_BUILD = "build-request";

    private DataBuilderService builderService = new DataBuilderService();

    protected void setup(){
        System.out.println("Iniciando Agent "+getAID().getName());

        DFAgentDescription dfad;
        ServiceDescription sd;

        dfad = new DFAgentDescription();
        sd = new ServiceDescription();

        dfad.setName(getAID());
        sd.setType("builder-agent");
        sd.setName("Builder-Agent");
        dfad.addServices(sd);

        try {
            DFService.register(this, dfad);
        } catch (FIPAException e) {
            System.out.println("Error registrando el servicio:
"+e.getMessage());
        }

        //Add the behaviour for taking in new building service requests
        addBehaviour(new BuilderRequestService());
        //Add the behavirous for do the build operation
        addBehaviour(new BuildOrderService());

    }
```

```
private class BuilderRequestService extends CyclicBehaviour{

        private static final long serialVersionUID = -3111362705889213177L;

        @Override
        public void action() {
            MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.CFP);
            ACLMessage msg = myAgent.receive(mt);
            if(msg != null){
                System.out.println("Query received "+msg.getContent());

                ACLMessage reply = msg.createReply();

                reply.setPerformative(ACLMessage.PROPOSE);

reply.setContent(String.valueOf(PerformanceSensorService.determinateCurrent
PerformanceAvailability()));

                myAgent.send(reply);
            }else{
                block();
            }
        }
    }

private class BuildOrderService extends CyclicBehaviour{

        private static final long serialVersionUID = 7866009589084269460L;

        @Override
        public void action() {
            MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
            ACLMessage msg = myAgent.receive(mt);

            if(msg != null){

                String data = msg.getContent();
                ACLMessage reply = msg.createReply();
                getBuilderService().buildRawData(data);
                reply.setPerformative(ACLMessage.INFORM);
            }else{
                block();
            }
        }

    }
```

Apéndice D. SourceCode - mas_cbr_app

```
package co.emgiraldo.cbr.similarityservices;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Set;
import co.emgiraldo.cbr.entity.KmVectorSensorAlerts;
import jcolibri.exception.NoApplicableSimilarityFunctionException;
import
jcolibri.method.retrieve.NNretrieval.similarity.LocalSimilarityFunction;

public class SetSimilarityFunction implements LocalSimilarityFunction{

    public double compute(Object arg0, Object arg1)
            throws NoApplicableSimilarityFunctionException {
        if ((arg0 == null) || (arg1 == null))
            return 0;
        if(!(arg0 instanceof Set))
            throw new
jcolibri.exception.NoApplicableSimilarityFunctionException(this.getClass(),
arg0.getClass());
        if(!(arg1 instanceof Set))
            throw new
jcolibri.exception.NoApplicableSimilarityFunctionException(this.getClass(),
arg1.getClass());

        Set<KmVectorSensorAlerts> hash0 = (Set<KmVectorSensorAlerts>)arg0;
        Set<KmVectorSensorAlerts> hash1 = (Set<KmVectorSensorAlerts>)arg1;

        Collection<KmVectorSensorAlerts> list1 = new ArrayList<>(
                Arrays.asList(hash0.toArray(new
KmVectorSensorAlerts[hash0.size()])));
        int size = list1.size();
        Collection<KmVectorSensorAlerts> list2 = new ArrayList<>(
                Arrays.asList(hash1.toArray(new
KmVectorSensorAlerts[hash1.size()])));

        list1.retainAll(list2);
        return Math.round((Double.valueOf(list1.size()) /
Double.valueOf(size)) * 100.0) / 100.0;
    }    public boolean isApplicable(Object o1, Object o2) {
        if((o1==null)&&(o2==null))
            return true;
        else if(o1==null)
            return o2 instanceof Set;
        else if(o2==null)
            return o1 instanceof Set;
        else
            return (o1 instanceof Set)&&(o2 instanceof Set);
    }
}
```

```
public class AttackCbr implements StandardCBRApplication{

    private Connector cbrConnector;
    CBRCaseBase cbrCaseBase;
    private VectorDataRules vectorDataRuleDao;
    private VectorData vectorDataDao;
    private SourceDataParser sourceDataParser;

    @Override
    public void configure() throws ExecutionException {
        cbrConnector = new DataBaseConnector();
        vectorDataDao = new VectorData();
        vectorDataRuleDao = new VectorDataRules();
        sourceDataParser = new SourceDataParser();

cbrConnector.initFromXMLfile(FileIO.findFile("/co/emgiraldo/cbr/config/data
baseconfig.xml"));
        cbrCaseBase = new LinealCaseBase();
    }

    @Override
    public void cycle(CBRQuery arg0) throws ExecutionException {}

    @Override
    public void postCycle() throws ExecutionException {
        // TODO Auto-generated method stub
        cbrConnector.close();
    }

    @Override
    public CBRCaseBase preCycle() throws ExecutionException {
        // TODO Auto-generated method stub
        cbrCaseBase.init(cbrConnector);
        Collection<CBRCase> cases = cbrCaseBase.getCases();
        for(CBRCase c:cases){
            System.out.println(c);
        }
        List<PairedAddressesVo> pairedAddresses =
getSourceDataParser().pairSourceAndDestAddresses(

getVectorDataDao().queryAddressesDistinct(VectorData.SOURCE_QUERY),

getVectorDataDao().queryAddressesDistinct(VectorData.DESTINATION_QUERY));

        return cbrCaseBase;
    }
```

```
public static void processCase(KmVectorData data,
List<KmVectorSensorAlerts> lista) {
        AttackCbr attackCbr = new AttackCbr();
        try{
            attackCbr.configure();
            attackCbr.preCycle();
            attackCbr.postCycle();




            Set<KmVectorSensorAlerts> tmp = new
HashSet<KmVectorSensorAlerts>(lista);

            data.setKmVectorSensorAlertses(tmp);

            CBRQuery query = new CBRQuery();
            query.setDescription(data);

            NNConfig config = new NNConfig();
            Attribute attribute;
            LocalSimilarityFunction function;

            attribute = new Attribute(VectorData.SRC_IP,
KmVectorData.class);
            config.addMapping(attribute, new StringSimilarityFunction());
            config.setWeight(attribute, 1.0);

            attribute = new Attribute(VectorData.DST_IP,
KmVectorData.class);
            config.addMapping(attribute,new StringSimilarityFunction());
            config.setWeight(attribute, 1.0);

            attribute = new Attribute("kmVectorSensorAlertses",
KmVectorData.class);
            config.addMapping(attribute, new SetSimilarityFunction());
            config.setWeight(attribute, 1.0);

            config.setDescriptionSimFunction(new Average());

            Collection<RetrievalResult> eval =
NNScoringMethod.evaluateSimilarity(attackCbr.cbrCaseBase.getCases(), query,
config);

            Collection<CBRCase> selectedCases =
SelectCases.selectTopK(eval, 5);
            List<CBRCase> caseToRetain = new ArrayList<>();
```

```
    for(CBRCase c:selectedCases){

                KmVectorData toRetain = (KmVectorData)c.getDescription();
                int p = new
Date().getSeconds()+attackCbr.cbrCaseBase.getCases().size();
               toRetain.setIdmasVectorData(p);
                caseToRetain.add(c);
                Iterator<KmVectorSensorAlerts> it =
((KmVectorData)c.getDescription()).getKmVectorSensorAlertses().iterator();
            }
            attackCbr.cbrCaseBase.learnCases(caseToRetain);
            attackCbr.postCycle();
```

Apéndice E. Evento Externo Payload

En la figura 73, se muestra la especificación del *payload* asociado a un evento específico en la plataforma *snorby*.

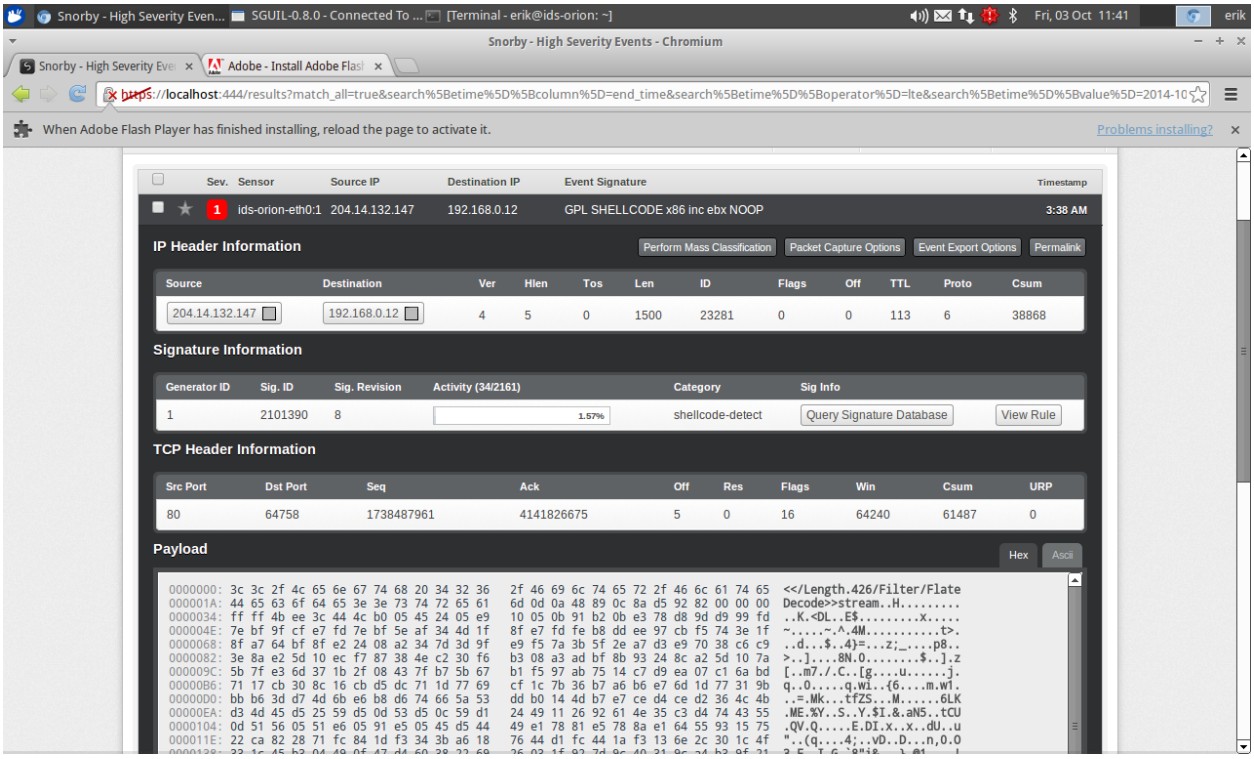

Figura 73. Payload de Evento en Snortby

Apéndice F. Detalles máquinas entorno controlado

F1. Máquina Vulnerable.

En la figura 74, se puede observar los componentes base del entorno vulnerable.

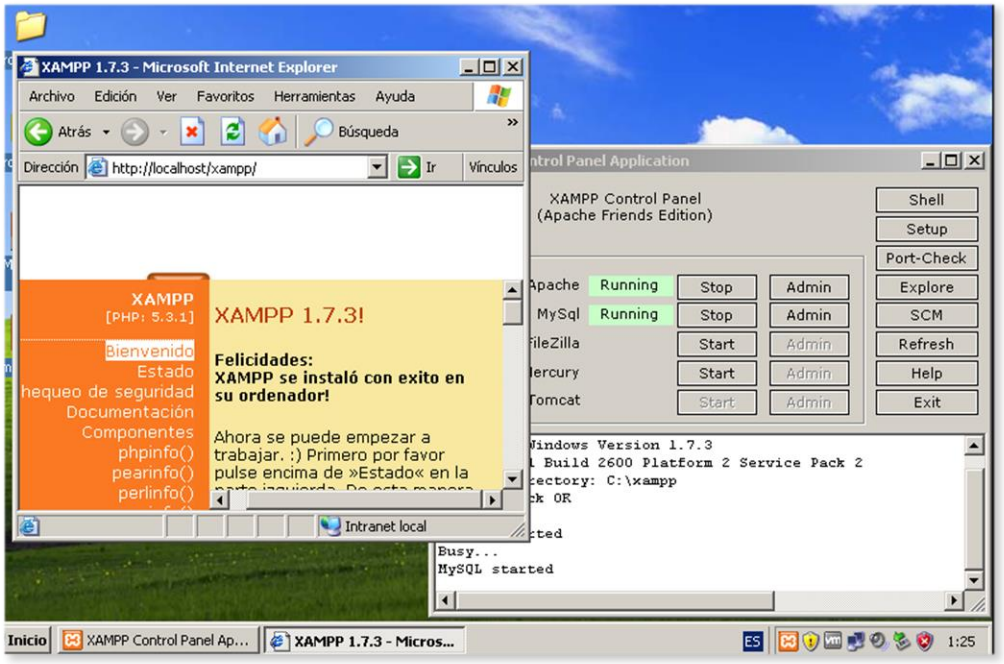

Figura 74. Servidor XAMPP y MySql en plataforma vulnerable

En la figura 75, Se observa la ejecución de wordpress vulnerable dentro de la plataforma XAMPP.

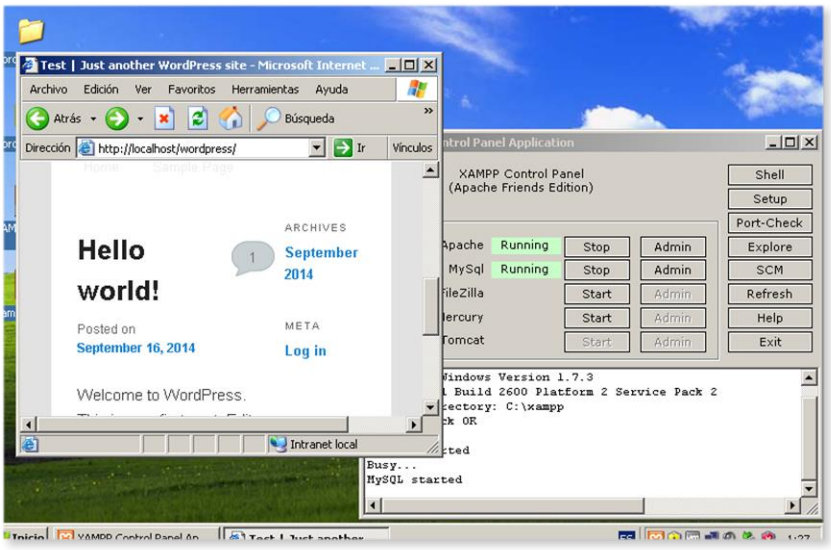

Figura 75. Wordpress Vulnerable en la Plataforma XAMPP

Apéndoce G. Entorno JADE

En la figura 76, se presenta la ejecución del agente *Builder* por consola.

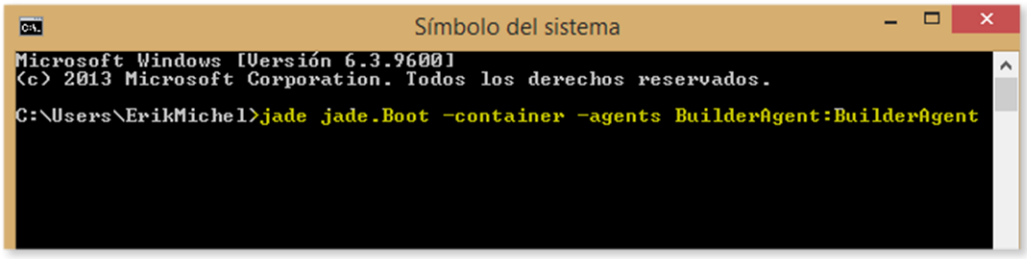

Figura 76. Agente *Builder* por Consola

En la figura 77. Se muestra la interfaz de administración de agentes.

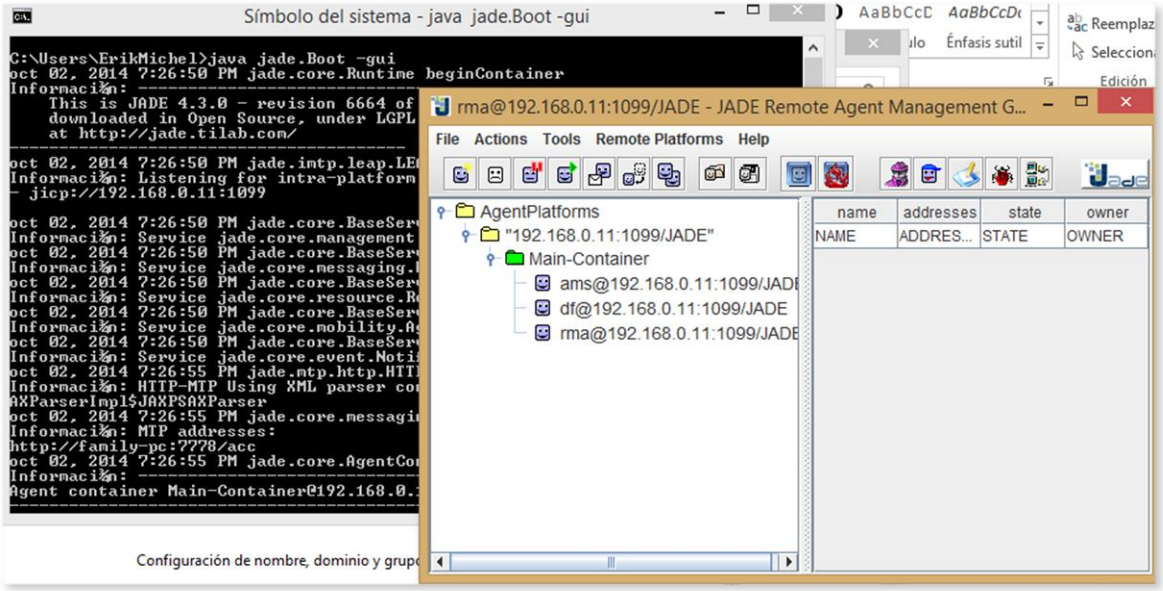

Figura 77. UI de Administración Agentes

Apéndice H. SourceCode - mas_data_core

H1. Agente Poller.

- **PSP0 Project Plan Summary**

Student	Erik Michel Giraldo Giraldo	Date	
Program	Sistema Multi-Agente deliberativo para la obtención y análisis de datos de *Honeynet*	Program #	1
Instructor		Language	Java

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning		42	42	33.33%
Design		31	31	24.60%
Code		26	26	20.63%
Compile		1	1	0.8%
Test		25	25	19.84%
Postmortem		1	1	0.8%
Total	24	126	126	100%

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0
Design	1	1	33.33%
Code	1	1	33.33%
Compile	1	1	33.33%
Test	0	0	0
Total Development	3	3	100%

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0
Design	0	0	0
Code	0	0	0
Compile	0	0	0
Test	3	3	100%
Total Development	3	3	100%
After Development			

- **PSP Time Recording Log**

Student	Erik Michel Giraldo Giraldo	Date	
Program	Agente Poller	Program #	1
Instructor		Language	Java

Project	Phase	Start Date and Time	Int. Time	Stop Date and Time	Delta Time	Comments
Agente Poller	Planning	18/09/2013	15	30/10/2013	42	Revisión y documentación del proyecto.
Agente Poller	Design	30/10/2013	15	30/11/2013	31	Análisis y diseño del requerimiento, se presentan interrupciones por trabajo.
Agente Poller	Code	30/11/2013	10	26/12/2013	26	Se realiza la implementación del agente acorde al comportamiento y diseño planteado.
Agente Poller	Compile	26/12/2013		26/12/2013		Compilación automática realizada a través del IDE
Agente Poller	Test	27/12/2013	10	20/01/2014	25	Pruebas del agente
Agente Poller	Postmortem	25/01/2014		25/01/2014		Se revisan los formatos de planeación y de registro de tiempos

- **PSP Defect Recording Log**

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

Student	Erik Michel Giraldo Giraldo	Date	
Program	Agente Poller	Program #	1
Instructor		Language	

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Agente Poller		1	90	Compile	Test	1hora	

Description: Se presentan inconvenientes en la configuración de ejecución del agente en el entorno

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Agente Poller		2	80	Design	Test	1hora	

Description: Se malinterpreto el protocolo de comunicación FIPA y se envió parámetro equivocado.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Agente Poller		3	80	Code	Test	1hora	

Description: El formato especificado para el JSON con la librería GSON no era el correcto.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

H.2 Agente Builder

- **PSP0 Project Plan Summary**

Student	Erik Michel Giraldo Giraldo	Date	
Program	Agente Builder	Program #	2
Instructor		Language	Java

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning		22	64	24.42%
Design		73	104	39.69%
Code		29	55	20.99%
Compile		1	2	0.76%
Test		10	35	13.35%
Postmortem		1	2	0.76%
Total	120	136	262	100%

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0%
Design	0	1	33.33%
Code	1	2	66.66%
Compile	0	0	0
Test	0	0	0
Total Development	1	3	100%

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0
Design	0	0	0
Code	0	0	0
Compile	0	0	0
Test	1	4	4
	1	4	100%
After Development			

- **PSP Time Recording Log**

Student	Erik Michel Giraldo Giraldo	Date	
Program	Agente Builder	Program #	2
Instructor		Language	Java

Project	Phase	Start Date and Time	Int. Time	Stop Date and Time	Delta Time	Comments
Agente Builder	Planning	25/01/2014	5	15/02/2014	22	Revisión y documentación del proyecto.
Agente Builder	Design	15/02/2014	30	28/04/2014	73	Análisis y diseño del requerimiento, se presentan interrupciones por trabajo.
Agente Builder	Code	28/04/2014	10	26/05/2014	29	Se realiza la implementación del agente acorde al comportamiento y diseño planteado.
Agente Builder	Compile	26/05/2014		26/05/2014		Compilación automática realizada a través del IDE
Agente Builder	Test	27/05/2014	10	15/06/2014	20	Pruebas del agente
Agente Builder	Postmortem	16/06/2014		16/06/2014		Se revisan los formatos de planeación y de registro de tiempos

- **PSP Defect Recording Log**

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

Student: Erik Michel Giraldo Giraldo
Date:
Program: Agente Builder
Program #: 2
Instructor:
Language:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
Agente Builder		1	100	Code	Test	4horas	

Description: Se presentan inconvenientes en la configuración del mapeo de las entidades de manera genérica diferentes a las propuestas para el cbr

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

H.3 Aplicación CBR (Case Base Reasoning).

- **PSP0 Project Plan Summary**

Student	Erik Michel Giraldo Giraldo	Date	
Program	CBR	Program #	3
Instructor		Language	Java

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning		31	95	23.05%
Design		14	118	28.64%
Code		29	133	32.28%
Compile		1	3	0.72%
Test		25	60	14.56%
Postmortem		1	3	0.72%
Total	15	101	412	100%

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0%
Design	0	0	0%
Code	2	4	100%
Compile	0	0	0%
Test	0	0	0%
Total Development	2	4	100%

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0%
Design	0	0	0%
Code	0	0	0%
Compile	0	0	0%
Test	2	6	100%
	2	6	100%
After Development			

- **PSP Time Recording Log**

Student	Erik Michel Giraldo Giraldo	Date	
Program	CBR	Program #	3
Instructor		Language	Java

Project	Phase	Start Date and Time	Int. Time	Stop Date and Time	Delta Time	Comments
CBR	Planning	16/06/2014	4	16/06/2014	31	Revisión y documentación del proyecto.
CBR	Design	16/06/2014	5	29/07/2014	14	Análisis y diseño del requerimiento, se presentan interrupciones por trabajo.
CBR	Code	29/07/2014	10	20/08/2014	29	Se realiza la implementación del agente acorde al comportamiento y diseño planteado.
Agente Builder	Compile	21/08/2014		21/08/2014	1	Compilación automática realizada a través del IDE
Agente Builder	Test	22/08/2014	10	15/09/2014	25	Pruebas del agente
Agente Builder	Postmortem	16/09/2014		16/09/2014	1	Se revisan los formatos de planeación y de registro de tiempos

- **PSP Defect Recording Log**

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

Student	Erik Michel Giraldo Giraldo	Date	
Program	CBR	Program #	3
Instructor		Language	

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
CBR		1	100	Code	Test	4horas	

Description: Se presentan inconvenientes en la configuración del mapeo de las entidades de manera específica diferentes a las propuestas para el cbr

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
CBR		2	80	Code	Test	2horas	

Description: Error en la codificación de la función de comparación asociada al arreglo de alertas asociadas

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

H.4 Front-End e Integración.

- **PSP0 Project Plan Summary**

Student	Erik Michel Giraldo Giraldo	Date	
Program	Front-end	Program #	4
Instructor		Language	Java

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning		1	96	22.11%
Design		1	119	27.41%
Code		15	148	34.10%
Compile		1	4	0.92%
Test		3	63	14.51%
Postmortem		1	4	0.92%
Total	20	22	434	100%

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0%
Design	0	0	0%
Code	0	4	100%
Compile	0	0	0%
Test	0	0	0%
Total Development	0	4	100%

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0%
Design	0	0	0%
Code	0	0	0%
Compile	0	0	0%
Test	0	6	100%
	0	6	100%
After Development			

- **PSP Time Recording Log**

Student	Erik Michel Giraldo Giraldo	Date	
Program	FrontEnd	Program #	4
Instructor		Language	Java

Project	Phase	Start Date and Time	Int. Time	Stop Date and Time	Delta Time	Comments
FrontEnd	Planning	16/09/2014		17/09/2014	1	Revisión y documentación del proyecto.
FrontEnd	Design	18/09/2014		19/09/2014	1	Análisis y diseño del requerimiento, se presentan interrupciones por trabajo.
FrontEnd	Code	19/09/2014	1	04/10/2014	15	Se realiza la implementación acorde al comportamiento y diseño planteado.
FrontEnd	Compile	04/10/2014		04/10/2014	1	Compilación automática realizada a través del IDE
FrontEnd	Test	04/10/2014		07/10/2014	3	Pruebas del UI
FrontEnd	Postmortem	08/10/2014		08/10/2014	1	Se revisan los formatos de planeación y de registro de tiempos

- **PSP Defect Recording Log**

Defect Types	
10 Documentation	60 Checking
20 Syntax	70 Data
30 Build, Package	80 Function
40 Assignment	90 System
50 Interface	100 Environment

Student Erik Michel Giraldo Giraldo Date
Program Front-End Program # 4
Instructor Language

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: Error en la codificación de la función de comparación asociada al arreglo de alertas asociadas

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description:

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.

Description: