Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación


# PROPUESTA DE ARQUITECTURA E IMPLEMENTACIÓN DE UN MÓDULO DE EJECUCIÓN DE ACCIONES DE RESPUESTA PARA UN SISTEMA AUTÓNOMO DE RESPUESTA A INTRUSIONES BASADO EN ONTOLOGÍAS

## TRABAJO FIN DE MÁSTER

**Danny Santiago Guamán Loachamín**

2013

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

**Máster Universitario en**
**Ingeniería de Redes y Servicios Telemáticos**

**TRABAJO FIN DE MÁSTER**

# PROPUESTA DE ARQUITECTURA E IMPLEMENTACIÓN DE UN MÓDULO DE EJECUCIÓN DE ACCIONES DE RESPUESTA PARA UN SISTEMA AUTÓNOMO DE RESPUESTA A INTRUSIONES BASADO EN ONTOLOGÍAS

Autor

**Danny Santiago Guamán Loachamín**

Director

**Víctor Abraham Villagrá González**

Departamento de Ingeniería de Sistemas Telemáticos

2013

## Resumen

En la actualidad, las tecnologías de la información a través de las redes de telecomunicaciones, terminales y una infinidad de aplicaciones y servicios han dado lugar a una sociedad cada vez más conectada. La oferta de servicios sobre la Internet permite que los usuarios puedan acceder a ellos desde cualquier lugar del mundo. No obstante, a la par de la consecución de esta sociedad de la información, Internet se ha convertido en un campo abierto en donde crackers, con diferentes objetivos, atentan contra la integridad, confidencialidad y disponibilidad de la información. En este sentido, el campo de la seguridad en redes ha requerido de una constante investigación para evaluar y optimizar los mecanismos de detección y control de acceso a diferentes recursos de la red. Dentro de estos mecanismos se encuentran los Sistemas de Respuesta a Intrusiones o IRSs, cuyo objetivo es reaccionar de forma automática, mediante la ejecución de una respuesta, ante la detección de una intrusión. Como parte de un proyecto de investigación emprendido por la Universidad Politécnica de Madrid se ha propuesto una arquitectura y se desarrolla un prototipo de un *Sistema Autónomo de Respuesta a Intrusiones basado en Ontologías,* cuyo objetivo es ejecutar una respuesta óptima de forma automática en función del contexto y del costo que implica ejecutarlo, a través de la evaluación de ciertas métricas.

Luego de evaluar el estado actual del AIRS basado en ontologías, enfocándonos en el módulo ejecutor de repuestas, se determinaron tres problemas de consideración: 1) Se permitía únicamente la ejecución local respuestas; no obstante una de los requerimientos de un IRS es su capacidad para ejecutar respuestas interactuando con componentes de seguridad de forma local y remota; 2) Escalabilidad limitada, lo cual dificulta el despliegue de nuevas respuestas de forma simple; 3) Dada la dificultad para el despliegue de nuevas acciones, el catálogo de acciones de respuesta del AIRS es reducido, lo cual dificulta su validación.

En este contexto, el presente Trabajo de Fin de Máster realiza la propuesta de arquitectura de un ejecutor de respuestas distribuido, seguro y escalable, que es adaptado al AIRS basado en ontologías. Además se lleva a cabo la implementación de de acciones de respuesta a modo de pruebas de concepto que son agregados al catálogo del AIRS.

La arquitectura del ejecutor de respuestas propuesto, está basado en plugins y lo constituyen seis componentes: *los IDSs*, se trabaja con NIDS Snort y HIDS OSSEC quienes se encargan de la detección de las intrusiones; *el razonador AIRS,* infiere la

respuesta óptima a través de la evaluación de varias métricas de respuesta; *el MCER (módulo central de ejecución de respuestas),* se encarga de construir una solicitud de respuesta y localizar a los agentes de ejecución; *el módulo de comunicación,* establece una conexión confiable y segura entre el MCER y los agentes de ejecución; *los agentes de ejecución,* llevan a cabo funciones de autenticación, listas blancas de ejecución, y la gestión de plugins; *y los componentes de seguridad,* que representan a los dispositivos que lleva a cabo la acción de respuesta real utilizando su interfaz de línea de comandos.

La utilización de múltiples IDSs distribuidos y de varios tipos, permite detectar un rango más amplio de intrusiones; además que si varios IDSs cubren un mismo conjunto de intrusiones, la confianza sobre las alertas emitidas desde estos IDSs aumenta, reduciendo los falsos positivos.

La provisión de agentes de ejecución distribuidos permite la ejecución de respuestas de forma local y remota, interactuando con diversos componentes de seguridad luego de que el razonador ha inferido la respuesta óptima. Así mismo, la arquitectura provee un marco común de comunicación y un conjunto de servicios independientemente de la respuesta a ejecutar; dicho marco permite establecer conexiones seguras y confiables entre el MCER y los agentes de ejecución, definir listas blancas, autenticar al MCER, deshacer una respuesta, entre otros. Por su parte, cada componente de seguridad posee sus propios medios de acceso y sintaxis de comandos para llevar a cabo una acción; por tal motivo la lógica de ejecución de una acción de respuesta es encapsulada en un plugin y se conecta al agente de ejecución a través de una interfaz de registro.

Finalmente mencionar que la arquitectura propuesta, al estar basada en plugins, permite un fácil despliegue de nuevas acciones de respuesta (protección, decepción, recuperación y pasivas) que interactúan con múltiples componentes de seguridad.

## Abstract

At present, the information technology through telecommunication networks, terminals, countless applications and services have led to an increasingly connected society. Services offerings on the Internet allows users to access them from anywhere in the world. However, with the achievement of this information society, Internet has become an open field where crackers, with different goals, threaten the integrity, confidentiality and availability of information. In this sense, the network security field has required constant research to evaluate and optimize the mechanisms of detection and control access to different network resources. Among these mechanisms are the intrusion response systems or IRSs, which aims is the automatic reaction by implementing a response to the intrusion detection. As part of a research project undertaken by the Universidad Politécnica de Madrid has proposed an architecture and developed a prototype of an Ontologies-Based Automated Intrusion Response System. Ontologies-Based AIRS aims to infer an optimal response to run automatically according to the context and the cost of running it, through the evaluation of certain metrics.

After evaluating the current state of AIRS based on ontologies, focusing on executing module responses, identified three issues for consideration: 1) only allowed local implementation responses, however one of the requirements of an IRS is its ability to run responses interacting with locally and remotely security components, 2) limited scalability, which makes difficult the deployment of new responses in a simple way, 3) AIRS catalog response actions is reduced, making it difficult to validate.

In this context, the present TFM proposes architecture of an executor of responses will be distributed, secure and scalable. This executor of responses is adapted to Based-Ontologies. It also performs implementation of response actions as a proof of concept that are added to the catalog AIRS.

The architecture of the proposed executor of responses consists of six components: *the IDSs*, we work with Snort NIDS and OSSEC HIDS which are responsible for detecting intrusions; *the AIRS reasoner* infers the optimal response by evaluating several metrics response; *the MCER* (central module execution responses), is responsible for building a request response and locate executor agents; *the communication module*, provides a reliable and secure connection between the MCER and implementing agents; implementing agents, execute functions of authentication, implementing whitelists, and the execution of the response action; and *security components*, which represent devices that performs the actual response action using their command line interface.

The use of multiple distributed IDSs allows to detect a wider range of intrusions. Furthermore, if multiple IDSs cover the same set of intrusions, then trust on alerts emitted by these IDSs increases and reducing false positives..

The provision distributed executor agents allows execution of responses locally and remotely and therefore can interact with various components of security, after the reasoner has inferred the optimal response. Also, the architecture provides a common framework for communication and a set of services regardless of the response to run. This framework allows: secure and reliable connections between the MCER and executor agents, define whitelists, authenticate the MCER, undo a response, and so on. In turn, each security component has its own access and command syntax to perform an action, for this reason the execution logic of a response action is encapsulated in a plugin and connects to agent execution through a register interface.

Finally mention that the proposed architecture is based on plugins and allows easy deployment of new response actions (protection, deception, recovery and passive) that interact with multiple security components.

# Índice general

# Índice de figuras

# Índice de Tablas

## Siglas

| | |
|---|---|
| IDS | Intrusion Detection System |
| IPS | Intrusion Prevention System |
| IRS | Intrusion Response System |
| ACL | Access List Control |
| NIDS | Network Based Intrusion Detection System |
| HIDS | Host Based Intrusion Detection System |
| AIDS | Application Intrusion Detection System |
| NNIDS | Node Network Intrusion Detection System. |
| NIPS | Network Based Intrusion Prevention System |
| HIPS | Host Based Intrusion Prevention System |
| IP | Internet Protocol |
| SID | Siganture ID |
| URL | Uniform Resource Locator |
| TAP | Test Access Port |
| ACK | Acknowledgment |
| ARP | Address Resolution Protocol |
| DDoS | Distributed Denial Of Service |
| DNS | Domain Name Service |
| DoS | Denial Of Service |
| IP | Internet Protocol |
| SYN | Synchronization |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |

# 1 Introducción

## 1.1 Contexto

Al día de hoy, sin lugar a duda hemos sido testigos de la espectacular evolución de las tecnologías de la información: las redes de telecomunicaciones, los terminales y una infinidad de aplicaciones y servicios han dado lugar a una sociedad cada vez más conectada. La mayor parte de la población cuenta con al menos un ordenador en casa o en la oficina, e incluso con teléfonos inteligentes, desde los cuales se accede a varias aplicaciones y servicios sobre la Internet. Así mismo, los sectores: industrial, bancario, militar, educativo, gubernamental y comercial; por mencionar algunos, ofertan cada vez más servicios sobre la Internet de tal forma que sus usuarios pueden acceder a ellos desde cualquier lugar del mundo.

Sin embargo, como se puede constatar en el Reporte de Amenazas de Seguridad de Internet del año 2011 emitido por Symantec [1], a la par de ésta evolución, Internet se ha convertido en el sitio preferido de crackers que constantemente y de forma creciente buscan vulnerabilidades (4.989 nuevas vulnerabilidades en el 2011) y las explotan utilizando herramientas y técnicas cada vez más sofisticadas y fáciles de usar; herramientas que permiten a los crackers crear nuevo malware y montar un ataque completo sin tener que escribirlo desde cero. En cualquier caso, las consecuencias de un ataque se traduce en un atentando contra la integridad, confidencialidad y disponibilidad de la información, ya sea que esté almacenada y sean accesibles desde Internet ó que transite a través de ella.

En éste sentido, el campo de la seguridad en redes ha requerido de una constante investigación para evaluar y optimizar los mecanismos de control de acceso y protección de la información en tránsito, que permitan mitigar los ataques. Tradicionalmente, se han empleado los firewalls y routers para aplicar una política de seguridad y definir listas de control de acceso respectivamente; sin embargo, estos dispositivos de defensa perimetral no pueden garantizar una protección del 100% contra los ataques existentes. Es así que surgen los Sistemas de Detección de Intrusiones (IDSs), como un nuevo mecanismo de defensa que permite detectar intrusiones o intentos de intrusiones que atenten contra la integridad, confidencialidad y disponibilidad de un recurso.

Los IDSs no son una tecnología nueva, el primer trabajo acerca de estos sistemas fue realizado por James Anderson en 1980 [2]; no obstante, esta tecnología desde sus inicios ha sido sujeta a una constante revisión y evolución. Es así que los *Sistemas de Detección de Intrusiones*, como menciona Anuar et al. en [3], además de ejecutar técnicas de detección llevan a cabo varias acciones de respuesta en contra de ellas, dando lugar

a los *Sistemas de Prevención de Intrusiones* (IPS) y a los *Sistemas de Respuesta a Intrusiones* (IRS).

Los tres sistemas tienen por objetivo monitorizar y detectar comportamientos anómalos suscitados en un sistema computacional y/o en la actividad de una red, sin embargo se diferencian en la forma en cómo reaccionan o responden ante un incidente. Un IDS es capaz de ejecutar el proceso de detección y tradicionalmente genera una advertencia o alerta hacia el administrador del sistema, para que sea él quien lleve a cabo una respuesta. El IPS por su parte, lleva a cabo el mismo proceso de detección que el IDS, pero su respuesta no se limita a una simple alerta, sino que es capaz de ejecutar una acción proactivamente para evitar que un ataque cumpla su objetivo, la respuesta normalmente consiste en acciones que bloquean el tráfico y lo llevan a cabo a través de un firewall o un dispositivo de control de acceso que debe estar ubicado *en línea* respecto al tráfico que se intenta filtrar [4]. El IRS también ejecuta una acción de respuesta, pero a deferencia del IPS sus acciones de respuesta no son ejecutadas necesariamente sobre dispositivos en línea con el flujo del tráfico y su acción es reactiva, es decir, si puede prevenir un ataque lo hará, pero su tarea más bien es reducir los daños causados por una intrusión. El IRS tiene por objetivo proveer un catálogo con varias acciones de respuesta, y la elección de una de ellas dependerá de un análisis previo que valore el costo que supone una intrusión respecto del costo de ejecutar una acción de respuesta, de éste modo se evita que las respuestas causen mayor daño que las propias intrusiones y además se minimiza el impacto del incidente [5].

En virtud de lo antes mencionado, se pueden identificar dos líneas de actuación claras, y que son sujetas de investigación actual: por un lado están las técnicas de detección de intrusiones, una revisión del estado del arte de estas técnicas se presenta en [6], [7] y [8]; y por otro están las técnicas de respuesta a intrusiones, línea en la cual se incluye el presente trabajo.

Los IRS han sido objeto de varias investigaciones desde hace algunos años: Stakhanova et al., presenta una taxonomía de los IRS junto con una revisión de las tendencias de investigación en la respuesta a intrusiones [9]; Anuar et al., hace un análisis y comparación entre 34 diferentes productos comerciales y no comerciales haciendo distinción entre las diferentes opciones de respuesta, y; Shameli-Sendi et al., presenta una revisión de los IRS clasificándolos de acuerdo a una taxonomía, en la que entre otros criterios se considera el método de selección de respuesta, la capacidad de adaptabilidad, el modelo de costo de respuesta y el tiempo de respuesta para clasificarlos.

Una de las investigaciones que van en la línea de los IRS que son capaces de adaptar la decisión de respuesta de forma automática en función del contexto y el costo que implica ejecutarlo, es el *Sistema Autónomo de Respuesta a Intrusiones basado en Ontologías* [10] [11].

Mateos et al. propone la arquitectura de un AIRS basado en ontologías, usando lenguajes formales de especificación de comportamiento y mecanismos de razonamiento, con el fin de garantizar coherencia semántica en un entorno heterogéneo, de tal forma que el AIRS tenga la capacidad de entender la sintaxis y semántica de las alertas de intrusiones generadas por diferentes IDS. El objetivo de la arquitectura propuesta, que se muestra en Figura 1, es ejecutar una respuesta óptima luego de evaluar ciertas métricas, como la severidad de una alerta, importancia de un host, etc., además de identificar si dos o más alertas se refieren a una misma intrusión o son diferentes.

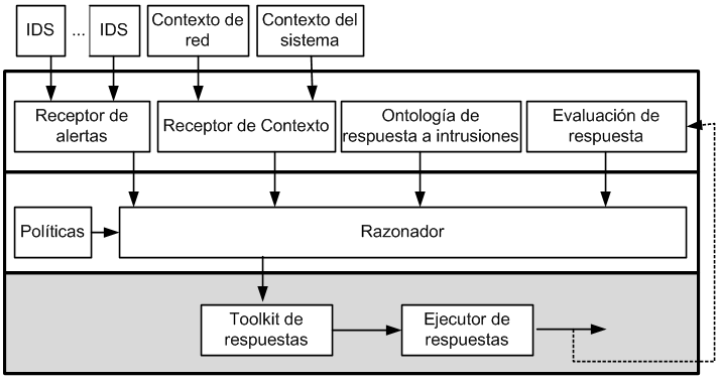



**Figura 1. Arquitectura del Sistema de Respuesta Automática a Intrusiones basado en Ontologías [10]**

Precisamente dentro de este contexto se enmarca el presente trabajo de fin de máster. El objetivo es contribuir al AIRS basado en ontologías, mediante una propuesta de arquitectura y posterior implementación del *módulo de ejecución de respuestas* (zona gris de la Figura 1) así como también con la implementación de pruebas de concepto de algunas *acciones* que serán agregadas al *toolkit de respuestas* para validar al AIRS basado en ontologías.

## 1.2 Objetivos

Los objetivos de este trabajo se pueden englobar en tres líneas de actuación claras:

- Revisar el estado del arte de:
  - Los Sistemas de Respuesta a Intrusiones (IRSs), y;
  - Las acciones de respuesta a intrusiones que pueden ser ejecutadas por un AIRS.

- Proponer e implementar una arquitectura del *módulo de ejecución de respuestas,* adaptable al AIRS basado en ontologías.

- Validación de la arquitectura propuesta mediante la implementación de pruebas de concepto de acciones de respuesta que permitan neutralizar ataques.

## 1.3 Metodología

Los objetivos anteriormente definidos marcan de forma genérica la estructura de la metodología a seguir durante el desarrollo del presente trabajo, pudiéndose definir las siguientes tareas:

**a) Análisis del estado del arte de los Sistemas de Respuesta a Intrusiones, y de las acciones de respuesta a intrusiones que pueden ser ejecutadas por un AIRS:**

**i.** La realización de las diferentes tareas del presente trabajo requiere del conocimiento previo de ciertos conceptos con los cuales se pueda trabajar, en este sentido se revisa los tópicos relacionados con los sistemas de detección de intrusiones, describiendo su arquitectura y taxonomía.

**ii.** Posteriormente se realiza una revisión del estado del arte de los sistemas de respuesta a intrusiones, especificando su taxonomía y las acciones de respuesta que puede ejecutar el AIRS en función del ataque. Así mismo se describe de forma detallada el AIRS basado en ontologías, propuesto por Mateos et al. [10], que sirve como base para este trabajo.

**b) Propuesta e implementación de una arquitectura del módulo ejecutor de respuestas, adaptable al AIRS basado en ontologías.**

**i.** Con la información obtenida en la tarea *ii)* del objetivo *a)*, se analiza el estado actual del módulo de ejecución de respuestas del AIRS basado en ontologías, se identifica sus limitaciones y se realiza el planteamiento del problema que intenta resolver el presente trabajo.

**ii.** Puesto que la acción o acciones de respuesta seleccionadas por el AIRS basado en ontologías, pueden requerir ejecutarse: sobre un firewall perimetral o personal, para agregar una regla de filtrado; sobre un router, para agregar una regla a la lista de control de acceso; sobre un servidor web, para restaurar archivos modificados; sobre un

host, para deshabilitar una cuenta de usuario que está siendo utilizada para efectuar un ataque, etc., se hace necesario entonces contar con un módulo de ejecución: distribuido, seguro, confiable y escalable. En este sentido, se realiza una propuesta de arquitectura y posterior implementación de un prototipo del *módulo de ejecución de respuestas* que se acople adecuadamente al AIRS basado en ontologías.

**iii.** Finalmente en base a la información obtenida en la tarea *ii)* del objetivo *a)*, se realiza la implementación de pruebas de concepto de las acciones de respuesta más representativas.

**c) Validación de la arquitectura propuesta mediante la implementación de pruebas de concepto de acciones de respuesta que permitan neutralizar ataques:**

**i.** Será necesario validar el prototipo desarrollado integrándolo con el AIRS basado en ontologías y realizar las pruebas respectivas para verificar y validar el funcionamiento del mismo.

## 1.4 Estructura de la memoria

La presente memoria intenta hacer una descripción de cada una de las tareas llevadas a cabo durante el proceso de desarrollo del presente trabajo de fin de máster, y que fueron definidas en la sección anterior. En este sentido, el contenido que sigue a este apartado está divido en 7 capítulos:

En el capítulo 2 se hace una revisión de los sistemas de detección de intrusiones, describiendo su arquitectura y su taxonomía. De la misma manera se hace una revisión del estado del arte de los sistemas de respuesta a intrusiones definiendo una taxonomía y las acciones de respuesta que pueden ejecutarse. Es importante mencionar que este capítulo no se limita a realizar una descripción de conceptos, sino que intenta hacer referencia a artículos científicos con alto índice de impacto que tratan diferentes tópicos de investigación actual relacionados con los sistemas de respuesta a intrusiones. Finalmente se describe el Sistema de Respuesta a Intrusiones Basado en Ontologías, que sirve de base para el presente trabajo.

El capítulo 3 aborda el problema en cuestión: en la primera parte, se describe el estado actual de desarrollo del módulo de ejecución de respuestas del AIRS basado en ontologías, y; en la segunda, se analizan los problemas que intenta dar solución el trabajo de fin de máster.

En el capítulo 4 se realiza el análisis de requerimientos para el módulo de ejecución de respuestas. En primera instancia se define un diagrama de casos de uso, en donde

establecen los requerimientos con un alto nivel de abstracción. Posteriormente se efectúa un esquema simple de la arquitectura, que brinde un marco de referencia para el establecimiento de requerimientos funcionales y no funcionales de cada componente identificado.

En el capítulo 5 se realiza la propuesta y diseño de arquitectura del módulo de ejecución de respuestas que cumpla con los requerimientos funcionales y no funcionales definidos en el capítulo 4. Luego de la presentación de 3 herramientas open source que llevan a cabo acciones de respuesta activas; como una decisión de diseño se opta por reutilizar algunos componentes de la herramienta open source SnortSam.

En el capítulo 6 se muestran los aspectos más representativos llevados a cabo en la implementación en la arquitectura diseñada en el capítulo 5. En primera instancia se muestra información del entorno de desarrollo empleado, y posteriormente se describe la implementación de cada módulo que conforma la arquitectura del ejecutor de respuestas.

En el capítulo 7 se presentan los resultados de las pruebas de validación realizadas en 3 escenarios: en el primer escenario, se muestra la ejecución de una respuesta activa de protección ante un ataque externo; en el segundo escenario, se realiza un ataque desde un host de la red interna de la organización, al cual el AIRS responde mediante una respuesta de protección que aisla al host del atacante; en el tercer y último escenario se muestran los resultados de la ejecución de una respuesta compuesta que involucra acciones de protección, decepción y una respuesta pasiva. En la sección final se realiza una evaluación de los resultados obtenidos.

Finalmente el capítulo 8, presenta las conclusiones y trabajos futuros que pueden desarrollarse tomando como base el presente trabajo.

# 2 Estado del arte

## 2.1 Sistemas de detección de intrusiones

Los sistemas de detección de intrusiones (IDSs), son sistemas de hardware y software que tienen por objetivo detectar ataques a medida que estos ocurren o después de que estos han tomado lugar. Dichos ataques, que son un conjunto de acciones no autorizadas o no deseadas, intentan comprometer la integridad, confidencialidad y disponibilidad de un recurso. Es decir, un IDS será capaz de detectar no solo un acceso no autorizado, sino también un intento de denegación de servicio o un escaneo de puertos por ejemplo. Esto es posible, ya que un IDS puede ser configurado para monitorizar el tráfico sobre una interfaz de red o actividades hostiles sobre un host; por consiguiente el IDS monitoriza, parsea y analiza: paquetes de red; llamadas al sistema; archivos de logs generados por routers, firewalls, servidores, sistemas locales, entre otros.

Esta sección describe la arquitectura básica de un sistema de detección de intrusiones y presenta una taxonomía del mismo. El objetivo es definir ciertos conceptos que permitan comprender de mejor manera los tópicos tratados en secciones posteriores.

### 2.1.1 Arquitectura básica

El Grupo de trabajo IDWG (Intrusion Detection Exchange Format) de la IETF define la arquitectura básica de un IDS, mostrada en la Figura 2 [12].

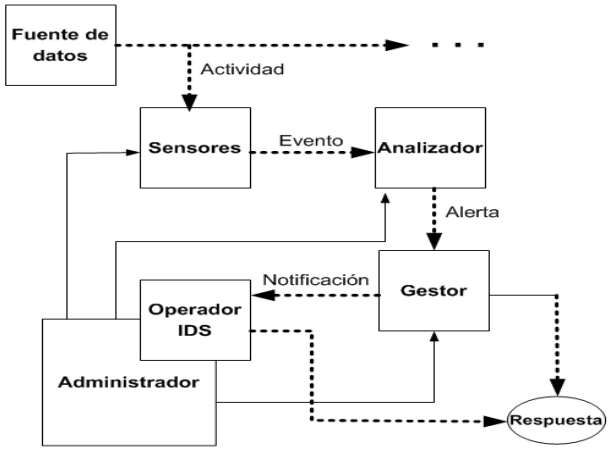



Figura 2. Arquitectura básica de un IDS.

***Administrador***

Es la persona encargada de establecimiento, despliegue y configuración de la política de seguridad de la organización; como es de esperar, dentro de esta política se

incluye al IDS. Un administrador no necesariamente es el *operador* del IDS, quien es el usuario principal del *gestor*. El *operador* juega un rol importante en la fase inicial de puesta en marcha del IDS, ya que será el encargado de establecer ciertos parámetros iniciales de los *sensores*, el *analizador* y el *gestor* del IDS. De igual manera, el operador a menudo monitoriza la salida del sistema de detección y de ser el caso lleva a cabo una acción de respuesta ante una intrusión detectada.

#### *Fuente de datos*

La fuente de datos es la información cruda que utiliza el IDS para detectar *actividades* no deseadas o no autorizadas. Paquetes que circulan a través de una interfaz de red, logs generados por el kernel del sistema operativo y sus aplicaciones, logs generados por firewalls o routers; son algunos ejemplos de fuentes de datos comunes. Las fuentes de datos son monitorizadas por los *sensores.*

#### *Sensores*

Los sensores son los encargados de recolectar información sobre la *fuente de datos* y posteriormente detectar ciertas *actividades* que serán enviadas como *eventos* hacia el analizador. El término *actividad* se refiere a los elementos u ocurrencias dentro una fuente de datos que son considerados de interés, y cuya monitorización ha sido definida por el *operador*. Intentos fallidos continuos de establecimientos de sesión telnet o ssh, modificación o copiado no autorizado del archivo passwd en los sistemas operativos Linux, peticiones de conexión recurrentes desde una misma dirección IP origen hacia el servidor web; son algunos ejemplos de las actividades que son detectadas por un sensor.

#### *Analizador*

Es un componente que analiza los datos de los *eventos* enviados desde los sensores buscando signos de actividades no autorizadas o no deseadas, o cualquier información de interés para el *administrador;* que son consideradas como una intrusión. Para llevar a cabo esta tarea, el analizador utiliza un *método de detección* que puede ser: *basado en uso indebido o basado en firmas, basado en anomalías, o un método híbrido* que combina los dos anteriores; una descripción de estos métodos se realiza en la sección 2.1.2.

Luego de que una intrusión ha sido detectada por el analizador, se envía un mensaje de *alerta* hacia el *gestor,* indicando que un evento de interés ha ocurrido.

#### *Alerta*

Una alerta, que es emitida desde el analizador hacia el gestor, contiene información tanto de la intrusión detectada como datos específicos de la ocurrencia en particular. El HIDS OSSEC, un IDS basado en host, genera ante un intento de ataque de Inyección SQL sobre PHP Nuke una alerta que entre otros datos contiene: dirección IP del atacante, fecha y hora del ataque, identificador de intrusión (SID), URL y navegador

utilizado [13]. Así mismo, el NIDS Snort, un IDS basado en red, genera ante una alerta que contiene: la numeración asociada a la descripción de la alerta, nombre de alerta, tipo de intrusión, prioridad, dirección IP del atacante, entre otros [14]. Como se puede apreciar, dependiendo del tipo de IDS y del tipo de intrusión el analizador genera una alerta con un formato en particular, incluso aún refiriéndose a una misma intrusión, diferentes IDSs pueden generar diferetes alertas; este problema es tratado por la IETF, quien define el Formato para el Intercambio de Mensajes de Detección de Intrusiones (IDMEF) [15]. Mateos et al. también trata el problema utilizando tecnologías de la web semántica en su AIRS basado en ontologías [10]; este trabajo es descrito en la sección 2.4.

#### *Gestor*

El gestor es un componente del IDS que recepta las alertas generadas por el analizador y toma la decisión de llevar a cabo una respuesta. Las respuestas, en algunos casos deben ser llevadas a cabo por el administrador u operador del IDS; otras por su parte, pueden ser ejecutadas automáticamente por alguna entidad de la arquitectura del IDS. Este componente es sujeto de varias investigaciones en la actualidad; la posibilidad de ejecutar acciones de respuesta de forma automática, es decir sin intervención del administrador, es una característica deseable luego de detectar una intrusión. Los sistemas de respuesta a intrusiones o IRS se caracterizan por ejecutar una acción de respuesta de forma automática; no obstante, escoger la acción o acciones de respuesta a ejecutar no es una tarea trivial y se ha convertido en un desafío en el diseño de los IRS. Es necesario evaluar el *costo* que implica ejecutar una acción de respuesta en todos los elementos de red [16]. Los IRSs son tratados con más detalle en la sección 2.2.

#### *Respuestas*

Las respuestas ejecutadas por un IDS, de acuerdo a la acción producida, pueden ser pasivas o activas [9]. Las *respuestas pasivas* no intentan minimizar el daño causado por un ataque, sino que su principal objetivo es enviar una *notificación* al administrador proveyéndolo de información sobre la intrusión. El envío de un correo electrónico, de un mensaje de texto, de una trap hacia una consola SNMP, o el registro de un log, son algunos ejemplos de respuestas pasivas.

Las *respuestas activas,* a diferencia de las anteriores, tienen por objetivo minimizar el daño hecho por el atacante. Ejemplos de este tipo de respuestas son: bloquear conexiones de red, deshabilitar una cuenta de usuario o terminar un proceso sospechoso, por mencionar algunos. Las acciones de respuesta son revisadas en la sección 2.3.

### 2.1.2 Taxonomía

| Trabajo | Enfoque | Tipos |
|---|---|---|
| **2011 Amer, S.H. et al. Intrusion Detection System Taxonomy – A short review** [17] | Fuente de auditoria | Archivos de logs; paquetes de red; archivos de logs de aplicaciones; alertas de sensores IDS;análisis de estado del sistema. |
| | Tecnología desplegada | Inalámbrica: fija y móvil; cableada. |
| | Procesamiento de datos | Distribuida; centralizada. |
| | Estructura o gestión | Distribuida; centralizada. |
| | Recolección de datos | Distribuida; centralizada. |
| | Tiempo de detección | Tiempo real; no en tiempo real. |
| | Sistema de preveción de intrusiones | |
| | Paradigma de detección | Basado en estado;basado en transición. |
| | Método de detección | Anomalías; uso indebido; híbrido; monitorización del objetivo; basado en especificación, escaner de vulnerabilidades; monitorización de integridad; monitorización de archivos de logs;honeypots;sondas secretas. |
| | Tipo de respuesta | Pasivo; activo. |
| | Sistema protegido o localización del IDS | Basado en red; basado en host; híbrido; basado en apliación. |
| | Frecuencia de uso o granularidad de datos | Contínuo; periódico |
| **2009 Villagrá, V. Seguridad en redes de telecomunicación** [18] | Fuentes de información que analiza | Basado en host; basado en red; basado en aplicación; basado en nodo de red. |
| | Frecuencia de tratamiento de eventos | Análisis periódico; monitorización contínua. |
| | Pirncipios de detección | Basado en firmas, basado en anomalías. |
| | Estrategia de control | Centralizada; ditribuida. |
| | Acciones de respuesta | Respuestas proactivas; respuestas reactivas: pasivas, activas. |

**Tabla 1. Taxonomía de los IDSs**

Varias taxonomías han sido propuestas con el objetivo de caracterizar de forma precisa a los sistemas de detección de intrusiones. Los trabajos más recientes corresponden a: Amer et al., quien presenta una taxonomía muy detallada clasificando a los IDSs desde 12 enfoques diferentes [17]; y Villagrá, quien realiza una clasificación genérica de los IDSs, considerando cinco enfoques [18].

El objetivo de esta sección no es describir cada uno de los IDSs señalados en la Tabla 1; este apartado toma como referencia los dos trabajos para describir a los IDSs desde cuatro enfoques diferentes*: según el sistema protegido o localización del IDS, según el método o principio de detección, según el despliegue de recolección de datos y estrategia de control, y según el tipo de respuesta*. Estos enfoques serán abordados ya que en el desarrollo del presente trabajo se utilizan o se hace referencia a IDSs que recaen en cada uno de estas categorías.

### *Según el sistema protegido o según la localización del IDS*

Anteriormente se señaló que la fuente de datos es aquella que provee los datos crudos a los sensores; por consiguiente, la localización de los sensores sobre una fuente de datos determina el sistema o entorno de red que monitoriza el IDS y da lugar a la siguiente clasificación: IDS basado en host [19] [20]; IDS basado en red [21] [22]; IDS basado en aplicación [18]; y los IDS basado en nodos de red [18].

#### Sistema de detección de intrusiones basado en host (HIDS)

El HIDS opera sobre un solo sistema; no obstante, puede monitorizar múltiples fuentes de datos:

- Sistema de archivos: Se verifica la integridad de los archivos y directorios, comparando información actual con información previamente recopilada (tamaño, dueño, fecha de modificación, por mencionar algunos).
- Registros de auditoría: son una colección de información sobre las actividades del sistema que han sido creadas cronológicamente y almacenados en ficheros. Existe un subsistema dedicado a generar registros de auditoría.
- Registros del sistema: son ficheros que almacenan los eventos generados por el sistema. En los sistemas Linux el daemon syslogd se encarga de generar y actualizar los registros de eventos suscitados.
- Registros de aplicaciones: son ficheros generados por aplicaciones que se ejcutan sobre un sistema; las bases de datos, firewalls y servidores web, son algunos ejemplos de aplicaciones que generan gran cantidad de registros.
- Conexiones establecidas con el sistema: el HIDS también puede detectar actividades relacionadas a ataques de red, por cuanto es capaz de interceptar todas las comunicaciones después de haber sido procesadas por

la capa de red y antes de pasar a los procesos a nivel de usuario. Esta característica le otorga al HIDS la capacidad de detectar intrusiones que pasan desapercibidas para los NIDS cuando la información viaja cifrada.

- Llamadas al sistema: la interacción entre las aplicaciones y el kernel del sistema operativo subyacente puede ser monitorizado mediante el rastreo de las llamadas al sistema.

### Sistema de detección de intrusiones basado en red (NIDS)

Un IDS basado en red tiene como fuente de datos el flujo de tráfico que circula a través de un segmento de red. El NIDS entonces, monitoriza los paquetes en busca de actividades que caracterizan a un ataque o intrusión; y que están dirigidas hacia los hosts que se encuentran en el segmento de red. El NIDS requiere que una interfaz de red trabaje en modo promiscuo, de esta manera todo el tráfico que circula a través del segmento de red será dirigido a capas superiores para su análisis. Un NIDS no debe interferir con el flujo de tráfico que monitoriza, por ello se suele utilizar los Puertos de Accesos de Pruebas (TAP) [23]. Los TAPs son dispositivos que capturan el tráfico en ambos sentidos y lo envía hacia el IDS sin interfir con el flujo de tráfico normal, su principal virtud radica en que el IDS conectado al TAP recibe el mismo tráfico que si estuviera en línea, incluyendo los errores.

### Sistema de detección de intrusiones basado en nodo de red (NNIDS)

El IDS basado en nodo de red, al igual que un NIDS, tiene como fuente de datos el flujo de tráfico que circula a través de un segmento de red; sin embargo, no tiene configurada una interfaz de red en modo promiscuo. El objetivo de un NNIDS no es monitorizar todos los nodos del segmento de red, sino que en su lugar monitorizan el tráfico dirigido desde o hacia un solo nodo en la red.

### Sistema de detección de instrusiones basado en aplicación (AIDS)

| IDS | Fortalezas | Debilidades |
|---|---|---|
| **HIDS** | -Puede analizar actividades sobre conexiones extremo a extremo cifradas.<br>-Capacidad para rastrear las llamadas del sistema. | -Falta de precisión en la detección debido a la falta de conocimiento del contexto.<br>-Retardos en la generación de alertas y reporte centralizado.<br>-Conflicto con controles de seguridad existentes.<br>-Gestión compleja en entornos con gran número de sistemas.<br>-Puede ser objetivo directo de un ataque. |
| **NIDS** | -Capacidad para analizar un alcance más amplio que los protocolos de aplicación.<br>-No afecta al rendimiento del resto de sistemas en la red.<br>-Puede ser indetectable, evitando ser objetivo directo de un ataque. | -Ataques sobre conexiones cifradas pasan desapercibidas.<br>-Alta tasa de falsos positivos y falsos negativos.<br>-Bajo alta carga de tráfico, no es posible la monitorización sobre todos los paquetes. |

**Tabla 2. Fortalezas y debilidades de los IDS según su localización o sistema protegido.**

El IDS basado en aplicación es una variación del HIDS, ya que en lugar de monitorizar las fuentes de datos de todo el sistema, monitoriza únicamente las de una aplicación específica. El AIDS está adaptado para una aplicación en particular; su uso es muy frecuente en la detección de intrusiones sobre servidores web y bases de datos.

En la Tabla 2 se hace una recopilación de las fortalezas y debilidades del HIDS y NIDS. En esta tabla no se incluyen los NNIDS y AIDS por cuanto son una especialización de las anteriores y acarrean sus mismas fortalezas y debilidades.

### *Según el principio o método de detección*

Una vez que se han definido las fuentes de datos y la localización de los sensores, éstos emiten eventos hacia el analizador. El analizador será el encargado de determinar si un evento corresponde a una intrusión, utilizando un método de detección: basado en anomalías [24]; o basado en firmas, [25] .

#### IDS basado en anomalías

La detección basada en anomalías define un perfil de comportamiento normal, ya sea de una red, de un host, de una aplicación o de perfiles de usuario. Cualquier variación de dicho comportamiento, más allá de ciertos umbrales, es detectada como una intrusión. Un HIDS que utiliza el principio de detección basado en anomalías monitoriza el comportamiento de las aplicaciones sobre el host observando la interacción que existe entre estas aplicaciones y el sistema operativo subyacente. El IDS rastrea las llamadas al sistema que realiza una aplicación "normalmente", e intenta aprender su comportamiento almacenándolo en una base de datos[1], entonces es posible reconocer un ataque cuando las llamadas al sistema no corresponden a su comportamiento normal [26].

Existen varias técnicas de detección basada en anomalías; Gyanchandani et al. define a la detección estadística de anomalías, detección basadas en minería de datos, detección basadas en conocimiento y detección basadas en aprendizaje automático; como los cuatro grandes grupos que abarcan a las diferentes técnicas de detección basada en anomalías [24].

#### IDS basado en usos indebidos

El IDS basado en usos indebidos o basado en firmas, almacena un conjunto de firmas que caracterizan a ataques conocidos; de tal forma que si existe una coincidencia entre los eventos analizados y estas firmas, se ha detectado una intrusión y se emite una alerta hacia el gestor IDS. La principal limitación de este método es que no puede detectar ataque nuevos cuyas firmas son desconocidas. Es decir que éste tipo de IDS

[1] En la fase de aprendizaje el HIDS rastrea las llamadas al sistema cuando éste no se encuentra bajo un ataque.

únicamente detecta ataques conocidos o ataques que no varíen mucho respecto de la firma almacenada. Algunas grupos de técnicas empleadas en éste método son: los sistema expertos, transición de estados, reglas de comparación y emparejamiento de patrones, y la detección basada en modelos.

| IDS | Fortalezas | Debilidades |
|---|---|---|
| **Basado en anomalías** | -Método efectivo para la detección de ataques desconocidos (ataques día cero).<br>-Menos dependiente del sistema operativo subyacente.<br>-Sistemas relativamente fáciles de mantener, no necesita actualizar una base de datos de firmas. | -Dificultad para definir perfiles de forma precisa, sobre todo en aquellos entornos en donde los eventos monitorizados cambian constantemente.<br>-La fase de aprendizaje puede durar días o semanas para que lleguen a ser efectivas.<br>-No esta disponible durante la fase de aprendizaje de comportamiento.<br>-Requieren un ajuste de umbrales precisos, de lo contrario si son demasiados altos se generan falsos negativos ó si es demasiado bajo muchos falsos positivos. |
| **Basado en usos indebidos** | -Método simple y efectivo para detectar ataques conocidos.<br>-Genera una tasa muy baja de falsos positivos. | -No es efectivo ante ataques desconocidos o variantes de ataques conocidos.<br>-Alto costo para mantener las firmas y patrones actualizados.<br>-Ante ataques nuevos, su tasa de falsos negativos es alta. |

**Tabla 3. Fortalezas y debilidades de los IDS según su método de detección.**

Finalmente mencionar que en la práctica, dependiendo del entorno, se emplean configuraciones híbridas aprovechando las ventajas de cada una, Aydin, M. et al. propone un IDS empleando ambos métodos [27].

### *Según la arquitectura de control*

En las secciones anteriores se ha clasificado a los IDS según la localización de la fuente de datos y según el método de detección, indicando sus fortalezas y debilidades; sin embargo, todos estos IDS tienen problemas de eficiencia. Cada IDS es especializado en detectar un conjunto específico de intrusiones, consecuentemente es probable que el mismo IDS no detecte otro conjunto de intrusiones; produciéndose falsos negativos. De la misma manera, algunos IDSs pueden emitir erróneamente alertas hacia el gestor IDS; produciéndose falsos positivos. Una solución para mejorar la eficiencia es usar IDSs distribuidos a modo de sondas locales. El utilizar múltiples IDSs permite detectar un rango más amplio de intrusiones, además como menciona Aussibal, J. et al. en [28], si varias sondas cubren un mismo conjunto de intrusiones, la confianza sobre las alertas emitidas desde estas sondas aumenta; reduciendo los falsos positivos. Ahora

bien, el despliegue distribuido de IDSs requiere de una estrategia de control que permita correlacionar las alertas emitidas de cada sonda. Existen dos métodos: centralizados [29] y distribuidos [28].

#### IDS basado en una arquitectura centralizada

El IDS basado en una estrategia centralizada está constituido por varias sondas que monitorizan varias fuentes de datos y que envían eventos hacia un único analizador que se encarga del proceso de correlación. Su principal fortaleza es que posee una vista consistente del contexto de red en un solo punto, pudiendo correlacionar las alertas y detectar ataques distribuidos. La principal desventaja de esta arquitectura centralizada es que existe un solo punto de fallo, pudiendo quedar totalmente inoperativa ya sea por un fallo o por sobrecarga.

#### IDS basado en una arquitectura distribuida

| IDS | Fortalezas | Debilidades |
|---|---|---|
| **Basado en una arquitectura centralizada** | -Posee una vista consistente de toda la red.<br>-Puede procesar alertas provenientes desde IDSs de diferentes tipos (HIDS, NIDS, IDS basados en anomalías o en usos indebidos).<br>-Disminución de la tasa de falsos positivos, si varias sondas cubren un mismo subconjunto de intrusiones. | -El proceso de correlación se concentra en un solo nodo, convirtiéndose en un solo punto de fallo. |
| **Basada en un arquitectura distribuida.** | -Tolerante a fallos.<br>-Capacidad para distribuir la carga del analizador, en condiciones de sobrecarga.<br>-Puede procesar alertas provenientes desde IDSs de diferentes tipos (HIDS, NIDS, IDS basados en anomalías o en usos indebidos).<br>-Disminución de la tasa de falsos positivos, si varias sondas cubren un mismo subconjunto de intrusiones. | -Mayor retardo en el intercambio de información de intrusiones detectadas. |

**Tabla 4. Fortalezas y debilidades de los IDS según su arquitectura.**

El IDS basado en una arquitectura distribuida está constituido por varias sondas y por varios analizadores que correlacionan las alertas. Cada analizador recibe alertas desde un subconjunto de sondas, y ejecuta el proceso de correlación por separado; de esta manera, no existe un solo punto de fallo sino que cada IDS es autónomo e independiente del resto de IDSs. Para detectar ataques distribuidos estos IDSs son capaces de intercambiar información sobre las intrusiones detectadas.

### *Según el tipo de respuesta*

Una vez que una intrusión o ataque ha sido detectado se emite una alerta desde el analizador hacia el gestor, que será el encargado de ejecutar una respuesta.

Dependiendo de su capacidad de respuesta, un IDS puede ser: basado en respuestas activas o basado en respuestas pasivas [9].

### IDS basado en respuestas pasivas

Un IDS con respuesta de acción pasiva únicamente tiene por objetivo el registro de la intrusión y no ejecuta acción alguna para evitar o mitigar los daños causados. En este caso, será el administrador o un tercer sistema el que lleve a cabo una respuesta, en caso de requerirlo. Dentro de las acciones de respuestas pasivas que puede ejecutar el IDS están:

- Notificación de la intrusión al administrador, mediante el envío de un correo electrónico o mensaje de texto.
- Envío de una trap SNMP hacia una consola de gestión.
- Registro de en un fichero de logs o en una base de datos de incidencias.

### IDS basado en respuestas activas

Un IDS basado en respuestas activas, además de registrar o notificar la intrusión ejecutando una de las acciones antes descritas; intenta mitigar el ataque o contrarrestarlo de forma autónoma, mediante la ejecución de una acción determinada sin requerir la intervención del administrador. Villagrá agrupa las respuestas activas en cuatro catergorías [18] :

- Respuestas de protección: Intentan anular cualquier tipo de interacción entre el atacante y la máquina de la víctima. Algunos ejemplos de este tipo de respuestas consisten en: agregar una política sobre un firewall para bloquear intentos de conexión; agregar reglas a las listas de control de acceso de un router; terminar los procesos asociados a conexiones con una dirección IP específica; bloquear un puerto; terminar un servicio y deshabilitar un usuario.
- Respuestas de recuperación: Intentan restaurar al recurso atacado a un estado anterior. Algunos ejemplos de estas respuestas son: restauración de los ficheros de un sitio web luego de haber sido sometido a un defacement, restauración de una BDD luego de que se han realizado cambios no autorizados, restauración de un host a un estado previo, entre otros. Dentro de este grupo también se incluyen aquellas acciones que ajustan la configuración de los equipos de detección.
- Respuestas de decepción: Su objetivo es simular sistemas vulnerables con el objetivo de atraer a los atacantes, de tal forma que se pueda recoger información del contexto del ataque: quién lo lleva a cabo, con qué objetivo y cuales son las técnicas empleadas. Los honeypots [30] y honeynets [31] son algunas tecnologías empleadas para desplegar respuestas de decepción.

- Respuestas de reacción: Tienen por objetivo ejecutar un contraataque. Sus fines pueden ser descubrir la identidad del atacante o encontrar información personal; no obstante, existen temas legales a tomar en cuenta, más aun cuando un ataque puede ser efectuado desde una jurisdicción diferente al de la víctima [32].

Ejecutar una acción de respuesta de forma automática no es una tarea trivial; es necesario evaluar el costo que implica ejecutarla, ya que inclusive podría ser más alto que aquel que causaría la intrusión en sí. Desde comienzos del siglo 21, se han realizado varias investigaciones respecto a la respuesta a intrusiones; en virtud de ello, la siguiente revisión presenta el estado del arte de los *sistemas de respuesta a intrusiones*.

## 2.2 Sistemas de respuesta a intrusiones (IRS)

Suele existir confusión respecto del modo en que operan los sistemas de respuesta a intrusiones (IRS) y los sistemas de prevención de intrusiones (IPS). Ambas tecnologías tiene en común la necesidad de un IDS, que será encargado de detectar una intrusión y notificarla para ejecutar una respuesta. La respuesta, que dicho sea de paso es de *tipo activa*, puede ser ejecutada por un IPS o un IRS.

Un IPS tiene por objetivo ejecutar una *respuesta de protección,* mediante: el bloqueo de potenciales intrusiones; o la terminación de conexiones, procesos y accesos asociados a una intrusión actual. Por tal motivo un IPS debe estar ubicado en línea con el flujo de transacciones que incluyen a la intrusión. Por ejemplo: un IPS que responde a intrusiones generadas por un NIDS debe estar ubicado sobre un firewall a través del cual fluye el tráfico de la potencial intrusión [33]; de la misma manera, un IPS que responde a intrusiones generadas por un HIDS que monitoriza las llamadas al sistema, debe estar ubicado necesariamente en el kernel del sistema operativo subyacente de tal forma que pueda bloquear las llamadas al sistema asociadas a una intrusión [34].

Un IRS cumple con las mismas funciones que un IPS; sin embargo, no se limita a ejecutar acciones de bloqueo sobre dispositivos en línea. Un IRS dispone de un conjunto de respuestas que se pueden ejecutar sobre otros componentes de seguridad; y la selección de una de ellas se hace en función de un análisis previo que evalúa el costo de ejecutar dicha acción. Otras características importantes en los IRSs son la capacidad de un ajuste adaptativo y un mecanismo de selección de respuesta dinámico. Shameli-Sendi et al. en [16], Stakhanova et al. en [9] , y Anuar et al. en [3] hacen una revisión del estado de los IRSs; una taxonomía de los IRSs es presentada por los dos primeros y se muestra en la figura 3.

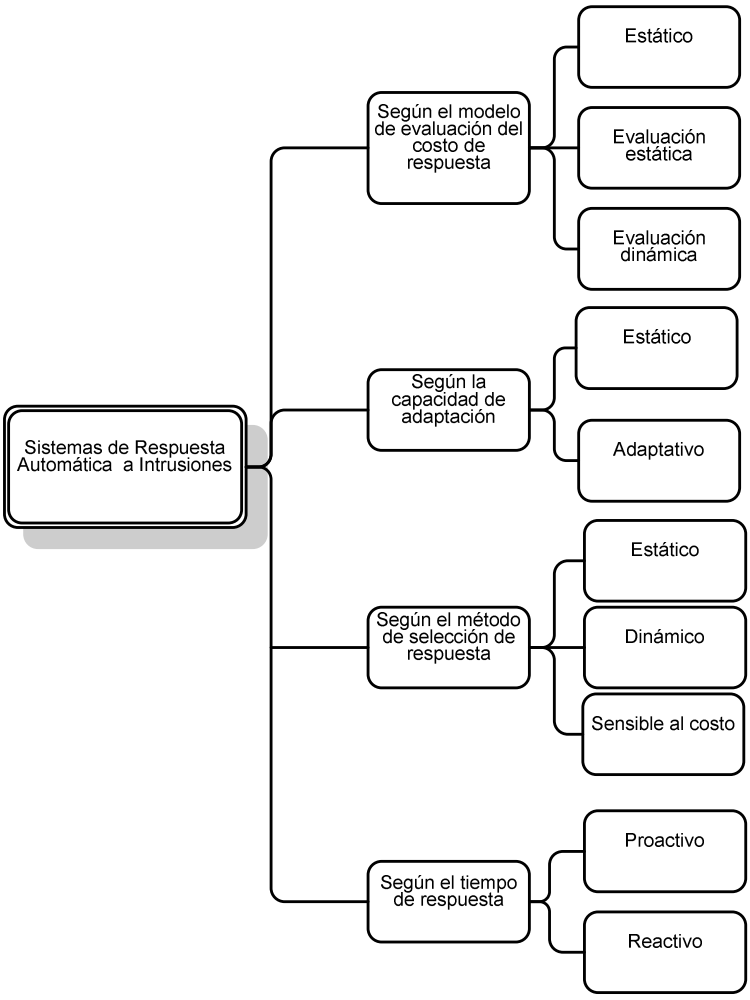



**Figura 3. Taxonomía de los Sistemas de Respuesta a Intrusiones [9] [16].**

### 2.2.1 Taxonomía

#### *Según el modelo de evaluación del costo de respuesta*

Considérese los siguientes grupos de ataques: denegación de servicio, escalamiento de privilegios, y acceso y control remoto no autorizado. Además se plantean dos posibles escenarios: escenario A, el proceso apache que mantiene un sitio web disponible a los usuarios ha sido secuestrado por un atacante; escenario B, se ha descubierto un bug sobre la aplicación skype que dispara el consumo de CPU considerablemente. En ambos casos el IRS ha seleccionado una respuesta activa, que consiste en terminar el proceso. Ahora bien, considerando que el objetivo de todo mecanismo de seguridad es garantizar la confidencialidad, integridad y disponibilidad de un recurso, se tiene que:

- Si en el escenario A, el objetivo del ataque era escalar privilegios y copiar el archivo passwd; entonces al ejecutar una acción de respuesta se incrementa

la confidencialidad y disponibilidad de un recurso. Pero el impacto negativo es la no disponibilidad del sitio web.

- Si en el escenario B, el objetivo del ataque era una denegación de servicio; entonces al ejecutar una acción de respuesta se incrementa la disponibilidad y no existe cambio alguno en la confidencialidad e integridad.

Como se puede apreciar, en ambos escenarios se tienen resultados diferentes pese a ejecutar una misma acción de respuesta. Por consiguiente es importante que al evaluar el costo que implica ejecutar una respuesta se considere el contexto sobre el que se ejecuta; por ejemplo considerando el tipo de ataque, la importancia del host, la localización del host, el número de usuarios actuales, disponibilidad de otras aplicaciones, etc.

Dependiendo del modelo empleado para evaluar el costo de ejecutar una respuesta, se tienen tres tipos: estático, evaluación estática y evaluación dinámica.

#### Estático

El costo de ejecutar una respuesta es obtenido mediante la asignación de un valor estático basado en la opinión de un experto. ADEPTS es un IRS que emplea este modelo [35].

#### Evaluación estática

El costo de ejecutar una respuesta no es dado por un experto; en su lugar, es obtenido estadísticamente evaluando los efectos positivos y negativos de ejecutarla. Los aspectos positivos están basados en métricas de disponibilidad, confidencialidad, integridad y rendimiento de cada respuesta. Mientras que los aspectos negativos pueden basarse en la disponibilidad y rendimiento de otros recursos luego de ejecutar la respuesta. Por ejemplo en el escenario A, anteriormente planteado, la respuesta que termina el servicio incrementa la integridad y confidencialidad, pero el efecto negativo es la indisponibilidad del servicio. Un IRS, cuya decisión está basada en una red jerárquica de tareas [36] y OrBAC [37], emplean un modelo de evaluación estática.

#### Evaluación dinámica

En este modelo el costo de ejecutar una respuesta es obtenido dinámicamente en función del contexto subyacente de ese momento. Por ejemplo en el escenario A, anteriormente planteado, la indisponibilidad del servicio luego de ejecutar la acción de respuesta, tendrá un costo diferente dependiendo del número de usuarios conectados al sitio web en ese momento. Si el costo es alto, entonces otra respuesta podría ser una mejor opción; quizá el filtrado de la dirección IP del atacante en el firewall de frontera. Así, este modelo evalúa el costo en función del contexto actual y cada vez que se

dispone a ejecutar una respuesta. Kheir et al. propone un IRS empleando una aproximación de este modelo en [38].

### *Según la capacidad de adaptación*

Una vez que el IRS ha seleccionado una respuesta, este puede evaluar o no el éxito del mismo con el fin de obtener un factor de efectividad. Este factor puede ser utilizado a su vez para autoajustar o adaptar una respuesta a determinada intrusión. Dependiendo de su capacidad de adaptación un IRS puede ser estático o adaptativo.

#### Estático

En este enfoque, el mecanismo de selección de respuesta permanece invariante ya que no existe un seguimiento de la respuesta desplegada. Por ejemplo, ante un ataque de denegación de servicio el IRS ha configurado la ejecución de una acción que bloquee la dirección IP del atacante añadiendo una regla de filtrado al firewall. Luego de ejecutar dicha respuesta, el IRS no hace un seguimiento para determinar si el ataque fue neutralizado. Ante continuas ejecuciones de respuestas fallidas o no efectivas, que dicho sea de paso deben ser identificadas por el administrador, se puede actualizar el mecanismo de selección de respuesta de forma manual; no obstante, hasta que ello ocurra varias intrusiones pudieron haber comprometido el recurso atacado. El IRS propuesto por Kheir en [38] posee este enfoque.

#### Adaptativo

El IRS adaptativo tiene la capacidad de ajustarse dinámicamente. Esta capacidad de adaptación puede ser llevada a cabo de varias formas; una aproximación para lograr este cometido es almacenar un historial de las respuestas ejecutadas. Lo que hace el IRS es almacenar el éxito (S) o no (F), de la ejecución de cada respuesta por cada host, manteniendo una métrica de efectividad (E) [5]. Ver Tabla 5.

| | **Host 1** | **Host 2** | **Host n** |
|---|---|---|---|
| **Respuesta 1** | $E_{1,1}=S_{1,1}-F_{1,1}$ | $E_{1,1}$ | $E_{1,n}$ |
| **Respuesta 2** | $E_{2,1}$ | $E_{1,2}$ | $E_{2,n}$ |
| **Respuesta m** | $E_{m,1}$ | $E_{m,2}$ | $E_{m,n}=S_{m,n}-F_{m,n}$ |

**Tabla 5. Historial de ejecuciones de respuesta que refleja la métrica de efectividad (E), en función de la ejecución exitosa (S) o fallida (F) de una respuesta sobre un host.**

Si la respuesta seleccionada neutraliza el ataque, se considera exitosa, la métrica de efectividad se incrementa en uno. Si la acción no ha neutralizado el ataque la métrica se decrementa. Este parámetro garantiza que el IRS sea adaptativo y pueda seleccionar la mejor respuesta en función de ejecuciones anteriores. El trabajo realizado por Stakhanova en [5] es un IRS de tipo adaptativo.

### *Según el método de selección de respuesta*

Dependiendo del método empleado por un IRS para seleccionar una respuesta, se tienen tres aproximaciones: mapeo estático, mapeo dinámico y mapeo dinámico sensible al costo.

#### Estático

Existe una correspondencia estática entre una intrusión y una acción de respuesta. Por ejemplo, como se observa en la Tabla 6; siempre que se detecte un ataque de denegación de servicio el IRS bloquea la dirección IP del atacante.

| Ataque | Respuesta |
|---|---|
| **Denegación de servicio** | Bloquear IP sobre firewall. |
| **Control remoto no autorizado mediante troyano** | Terminar proceso. |
| **Escalamiento de privilegios** | Deshabilitar usuario |

**Tabla 6. Asignación estática ataque-respuesta**

Estos sistemas son fáciles de construir y mantener; sin embargo, presenta dos problemas en particular:

- Son predecibles y por consiguiente vulnerables. Por ejemplo si el atacante realiza un ataque de DoS Smurf, y el IRS bloquea las direcciones IP; realmente está bloqueando las direcciones IP de su organización y no la del atacante. Causando un ataque de DoS a nivel interno.
- El IRS es incapaz de tomar en cuenta el contexto actual del sistema completo, las acciones de repuesta son esfuerzos aislados que intentan mitigar el ataque sin considerar las condiciones actuales del sistema y el impacto que causa sobre ella.

FLIPS es un IRS que emplea un mapeo estático [39].

#### Dinámico

Los IRS de mapeo dinámico asocian una alerta de intrusión a un conjunto de posibles respuestas. La selección de la mejor respuesta de entre este conjunto puede estar basada en múltiples factores, como: el estado actual del sistema, métricas de la alerta de intrusión, o una política de red.

| Ataque: Denegación de servicio | Métricas: Confianza (c), Severidad (s) |
|---|---|
| **Notificación** | si (c>0.3 && s>0.5) |
| **Bloquear IP sobre firewall.** | si (c>0.7 \|\| s>0.7) |
| **Desplegar honeynet y redirigir** | si (c>0.7 && s>0.5) |

**Tabla 7. Selección dinámica, en función de la confianza (c) y severidad (d) de la alerta de intrusión**

En la Tabla 7, ante un ataque de DoS se puede ejecutar tres acciones de respuesta. La selección en tiempo real de una de ellas, está determinada por las métricas asociadas a cada alerta de intrusión: confianza y severidad. Este enfoque provee mayor flexibilidad que el mapeo estático, ya que se pueden ajustar las métricas y el conjunto

de respuestas asociadas a cada intrusión. Por ejemplo, según la Tabla 30 aquellas alertas de intrusión con confianza menor a 0.3 y severidad menor a 0.5 son ignorados; no obstante, estos valores pueden ser ajustados por el administrador según su conveniencia. El AAIRS propuesto por Carver en [40] se incluye dentro de este tipo de IRSs.

#### Sensible al costo

Este es un enfoque que tiene por objetivo encontrar un balance entre el costo de ejecutar una respuesta y el costo de los daños causados por una intrusión. Tres opciones para determinar el costo de una respuesta fueron abordadas al empezar esta sección. En lo que se refiere al costo de los daños causados por una intrusión, algunas propuestas incluyen un componente de evaluación de riesgos que calcule un factor de riesgo para todos los recursos [16]. Hay dos tipos de componentes para la evaluación de riesgos: estático, en donde se asigna un factor de riesgo a cada recurso en la red y es obtenido a priori, utilizando alguna metodología estándar como los definidos por el NIST [41] y la ISO/IEC 27005 [42], y; dinámico, que consiste en un proceso de evaluación de riesgo en tiempo real de un recurso [43]. En ambos casos, el objetivo de hacer una evaluación de riesgos es evitar que el costo de ejecutar una respuesta sea mayor que el costo de los daños causados por la intrusión.

Los IRS propuestos por Kheir et al. en [38], Kanoun et al. en [37] y Stakhanova et al. en [5] proponen varios enfoques de selección de respuesta sensibles al costo.

### *Según el tiempo de respuesta*

Un componente esencial de los sistemas de respuesta a intrusiones son las respuestas propiamente dichas. En la sección 2.1.2 se clasifico a las respuestas en activas y pasivas dependiendo de su capacidad de respuesta ante un incidente. De acuerdo a la taxonomía antes mencionada, los IRSs de respuesta activa han sido extendidos en función del tiempo al que ejecutan las respuestas, pudiendo ser: proactivos o reactivos.

#### Reactivo

Un IRS reactivo ejecuta una acción de respuesta luego de que una intrusión ha sido detectada y confirmada. Dicha confirmación puede basarse en métricas de confianza de un IDS o en una coincidencia completa del vector de ataque con la firma definida en la BDD de un IDS [9]. Por lo general un IRS reactivo intenta minimizar el daño causado por una intrusión o restaurar el estado del sistema[2]. Este tipo de respuestas no son adecuadas para sistemas críticos, ya que es posible un escenario donde la ejecución de la respuesta ocurre luego de que el atacante ha cumplido su objetivo y no pueda

[2] Una clasificación de los tipos de respuestas activas se presenta en la sección 2.1.2.

restaurar fácilmente al sistema a un estado anterior. Jahnke et al. incluye en su trabajo un mecanismo de respuesta reactivo [44].

### Proactivo

En el IRS reactivo el atacante tiene una ventana de tiempo desde que inicia la actividad maliciosa hasta cuando se ejecuta una respuesta. Un IRS proactivo prevé la ocurrencia de una intrusión antes de que el ataque haya afectado al recurso objetivo, intentado controlarlo y prevenirlo mediante la ejecución de una respuesta. Varias técnicas son empleadas para prever una intrusión, algunas se basan aproximaciones probabilísticas, análisis del comportamiento del usuario, contexto actual del sistema, por mencionar algunos. Schultz et al. [45] propone un método de predicción para prevenir nuevos ataques. El IRS ADEPTS, propuesto por Foo et al. también presenta un enfoque basado en respuestas proactivas [35].

En la Figura 4, se puede apreciar la relación existente entre una respuesta pasiva y activa (reactiva y proactiva) representados sobre el marco temporal de un ataque o intrusión. Al tiempo $\mathbf{t_0}$ el sistema se encuentra en estado normal; el tiempo $\mathbf{t_1}$ denota el instante al que una alerta de intrusión es reportada; y el tiempo $\mathbf{t_2}$ denota el estado del sistema después de que una respuesta reactiva se ha ejecutado. En función de estos tiempos sobre el marco temporal del ataque, se tienen tres intervalos[3]:

- Tiempo **t** antes de la alerta de intrusión ($\mathbf{t_0 < t < t_1}$): En este intervalo de tiempo la ejecución de una respuesta proactiva juega un rol importante previniendo un ataque sobre un sistema de la red. Acciones de bloqueo y ajustes de configuración de sistemas de seguridad pueden ser desplegadas en esta fase de ejecución proactiva.
- Tiempo **t** después de la alerta de intrusión ($\mathbf{t_1 < t < t_2}$): En este intervalo de tiempo se ejecuta una respuesta reactiva cuyo objetivo es minimizar el impacto causado por la intrusión. Acciones de bloqueo de tráfico, terminación de procesos, deshabilitación de usuarios, etc., pueden ser ejecutados. Para el despliegue de estas respuestas puede ser necesario interactuar con otros componentes de seguridad como: firewalls, routers, aplicaciones sobre hosts, entre otros. Esta fase de ejecución reactiva termina al tiempo $t_2$.
- Tiempo **t** después de la ejecución de la respuesta reactiva ($\mathbf{t_2 < t}$): Este intervalo no tiene tiempo de finalización, ya que se trata de una fase de investigación. En esta fase el IRS evalúa y aprende de lo ocurrido en el incidente antes y después de la ejecución de la respuesta, el resultado sirve como retroalimentación para futuras respuestas. Información incluida en la

notificación de una respuesta pasiva puede servir también como retroalimentación.

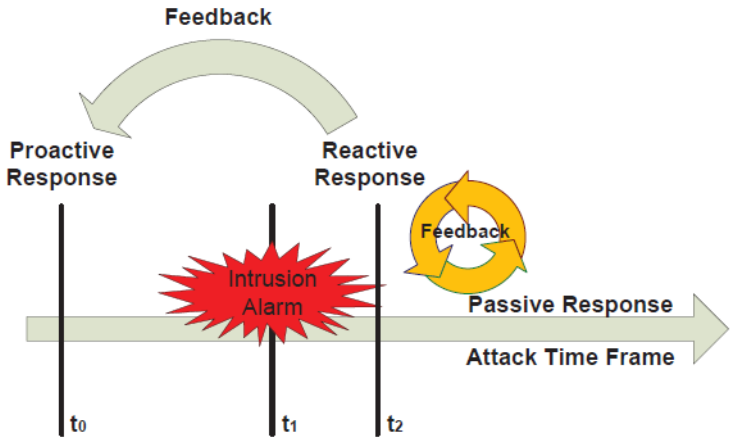



**Figura 4. Relación entre una respuesta pasiva, proactiva y reactiva sobre el marco temporal de un ataque [3].**

La siguiente sección presenta una clasificación detallada de las *acciones de respuesta* que puede ejecutar un IRS en los intervalos antes mencionados.

## 2.3 Acciones de respuesta de un IRS

En la sección 2.1.2 se presenta una clasificación genérica de las respuestas según su tipo: *pasivas y activas*; las activas a su vez pueden ser *de protección, recuperación, decepción o reacción (contraataque)*. De la misma manera en la sección 2.2.1, se realiza una extensión de las respuestas activas dividiéndola en dos subconjuntos según el tiempo al que son ejecutadas, pudiendo ser: *proactivas o reactivas*. No obstante, es importante conocer y entender las diferentes opciones de *acciones de respuesta* existentes; esto con el fin de realizar una asignación y priorización adecuada de las acciones de respuesta que puede ejecutar un IRS ante diferentes tipos de intrusiones.

En este contexto la presente sección aborda el tema relacionado a las acciones de respuesta, clasificándolas en función del tipo de ataque al que pueden responder. Para realizar esta clasificación se requiere de dos componentes: 1) Un conjunto de respuestas activas y pasivas que pueden ser ejecutados por el IRS a través de diferentes componentes de seguridad (routers, firewalls, IDSs, etc.) o a través de aplicaciones del sistema operativo subyacente (tcpwrapper, firewall local, comandos locales, etc.), y; 2) Una taxonomía que agrupe a las intrusiones en función del vector de ataque utilizado para llevarlo a cabo.

### 2.3.1 Acciones de respuesta

Con respecto al primer componente, intentar definir un conjunto de acciones de respuesta completo sería muy complejo. Además la definición de dicho conjunto de respuestas depende de la organización en la cual se despliegue. Un análisis de riesgos brindaría una buena aproximación de los recursos que se deben proteger sobremanera; por consiguiente, se tendría un panorama más claro respecto de qué acciones de respuestas se deben implementar. No obstante, Stanakhova et al. presenta en [9], una lista de acciones de respuestas comúnmente implementadas por los IRSs que se abordan en ese trabajo. Así mismo, Rash et al. [34] define las acciones de respuesta clasificándolas en: respuestas basadas en red, que son aquellas que tienen la capacidad de interactuar de forma directa con el tráfico de red, y en; respuestas basadas en hosts, que tienen la capacidad de interactuar sobre los procesos, servicios, conexiones y usuarios que se ejecutan sobre el mismo host.

| ID | Acción de respuesta | Pv | Pr | P | D | R | C |
|---|---|---|---|---|---|---|---|
| R-1 | - Notificación al administrador | √ | | | | | |
| R-2 | - Generación de reporte | | | | | | |
| R-3 | - Habilitación de herramientas de análisis adicionales | √ | √ | | | √ | |
| R-4 | - Respaldar archivos manipulados | | | | | | |
| R-5 | - Rastreo de la conexión del atacante para recopilar información | √ | | | | | √ |
| | **Basadas en host** | | | | | | |
| R-6 | - Denegación de acceso selectiva o completa a un archivo. | | | √ | | | |
| R-7 | - Eliminación de un archivo modificado. | | | √ | | | |
| R-8 | - Permitir la manipulación de un archivo falso. | | | | √ | | |
| R-9 | - Restauración de un archivo manipulado con un archivo de respaldo. | | | | | √ | |
| R-10 | - Restringir actividad del usuario | | | √ | | | |
| R-11 | - Deshabilitación de cuenta de usuario | | | √ | | | |
| R-12 | - Apagar el host comprometido | | | √ | | | |
| R-13 | - Apagar el servicio comprometido | | | √ | | | |
| R-14 | - Reiniciar proceso sospechoso | | | √ | | √ | |
| R-15 | - Bloquear llamadas al sistema sospechosas. | | | √ | | | |
| R-16 | - Retardar llamadas al sistema sospechosas. | | | √ | | | |
| R-17 | - Terminar proceso sospechosos | | | √ | | | |
| | **Basadas en red** | | | | | | |
| R-18 | - Agregar o borrar reglas de filtrado sobre un firewall | √ | | √ | | | |
| R-19 | - Reiniciar sistema atacado | | | √ | | √ | |
| R-20 | - Bloquear conexiones de red entrantes y salientes sospechosas | √ | | √ | | | |
| R-21 | - Bloquear puertos o direcciones IP | √ | | √ | | | |
| R-22 | - Despliegue de un host o red señuelo (honeypot o honeynet). | √ | | | √ | | |
| R-23 | - Suplantación de mensajes TCP RST o ICMP Unreachable para terminar conexiones TCP y UDP respectivamente. | | | √ | | | |

**Tabla 8. Acciones de respuesta. (Pv) Respuesta pasiva; (Pr) Respuesta proactiva; P (Respuesta activa protección); D (Respuesta reactiva de decepción); R (Respuesta reactiva de recuperación); C (Respuesta reactiva de reacción o contraataque).**

La Tabla 8, muestra una lista general de las acciones de respuesta propuestas en los antes mencionados trabajos. En esta tabla se realiza una clasificación de las acciones de respuesta basadas en red y basadas en host, indicando su tipo. Además se ha añadido un identificador (ID) a cada respuesta para hacerlo referencia en una sección posterior.

### 2.3.2 Taxonomía de ataques

Como se mencionó anteriormente, luego de definir las acciones de respuesta se requiere una clasificación de las intrusiones agrupadas por vector de ataque. Para el efecto se ha considerado la taxonomía propuesta por Hasman et al. en [46].

La taxonomía de Hansman establece cuatro dimensiones para categorizar un ataque de la forma más exacta posible, las dos primeras dimensiones pueden contener niveles adicionales dependiendo del grado de especialización requerido.

#### *Primera dimensión: vector de ataque*

Cubre el vector del ataque empleado y es utilizada para clasificar al ataque dentro de una clase. Varios niveles pueden ser definidos en esta dimensión con el fin de precisar su tipo; por ejemplo, un ataque de red puede ser un ataque web y este a su vez puede ser del tipo Inyección SQL (ver Tabla 9).

Las clases y niveles definidos en la Tabla 9, pueden servir como punto de partida para realizar una clasificación más exacta (extendida o reducida) acorde a las necesidades requeridas. Se definen nueve clases principales de ataques según el vector de ataque principal. Cuando una alerta de intrusión es generada, el clasificador debe categorizarla en una de estas nueve opciones:

- Virus: programa que se replica a sí mismo y que se propaga a través de archivos infectados. Una vez cargado en memoria cumple las instrucciones programadas por su creador, que pueden ser: destruir ficheros, modificar ficheros, sobrecargar recursos, entre otros.
- Gusanos: programa que se replica a sí mismo y que no requiere de un fichero infectado para propagarse, en su lugar se propaga a través de servicios de red o a través del correo electrónico.
- Troyanos: se ejecuta como un programa convencional; no obstante, normalmente es utilizado para permitir una conexión remota.
- Desbordamiento de buffer: procedimiento mediante el cual los datos que se escriben sobre un buffer corrompen aquellos datos en direcciones adyacentes de memoria, ganando el control de un proceso determinado.

| ID | Nivel 1 | ID | Nivel 2 | ID | Nivel3 |
|---|---|---|---|---|---|
| A1 | Virus | 1 | Infección de archivos | | |
| | | 2 | Infección de registros del sistema | | |
| A2 | Gusanos | 1 | Envío de correo masivo | | |
| | | 2 | Propagación red | | |
| A3 | Troyano | | | | |
| A3 | Desbordamiento de buffer | 1 | Stack | | |
| | | 2 | Heap | | |
| A4 | Denegación de servicio | 1 | Basado en host | 1 | Consumo de recursos |
| | | | | 2 | Bloqueo |
| | | 2 | Basado en red | 1 | Inundación UDP |
| | | | | 2 | Inundación TCP |
| | | | | 3 | Inundación ICMP |
| | | 3 | Distribuido | | |
| A5 | Ataques de red | 1 | Spoofing | 1 | IP Spoofing |
| | | | | 2 | DNS Spoofing |
| | | | | 3 | Web Spoofing |
| | | | | 4 | TCP Spoofing |
| | | | | 5 | ARP Spoofing |
| | | 2 | Secuestro de sesión (hijacking) | | |
| | | 3 | Ataques inalámbricos | 1 | Cracking WEP |
| | | 4 | Ataques web | 2 | XSS (Cross Site Scripting) |
| | | | | 3 | SQL Injection |
| | | | | | Ataques a BDD |
| A6 | Ataques físicos | 1 | Básico | | |
| | | 2 | Guerra de energía | | |
| A7 | Ataques de contraseña | 1 | Adivinación | 1 | Fuerza Bruta |
| | | | | 2 | Ataque de diccionario |
| A8 | Ataques de recolección de información | 1 | Sniffing | | |
| | | 2 | Mapping | | |
| | | 3 | Escaneo | | |

**Tabla 9. Categorías de primera dimensión de la taxonomía de ataques [46].**

- Denegación de servicio: evita que un usuario legítimo pueda hacer uso de un recurso. Para ello, satura los recursos del sistema de la víctima inhabilitando los servicios prestados por dicho sistema por un tiempo indefinido. Los recursos saturados normalmente son: computacionales, como la memoria, espacio en disco y procesador; y de red, que se traduce en un consumo de ancho de banda produciendo una pérdida de conectividad de los usuarios. Existen varios tipos de ataques de DoS: basados en hosts, basados en red y distribuidos.
- Ataques de redes: atentan contra las redes mediante la manipulación protocolos pertenecientes al stack de protocolos TCP/IP. Su objetivo es introducirse en un sistema suplantando la identidad de un usuario o del propio administrador.
- Ataques físicos: su objetivo es causar daño a los componentes físicos de una red.

- Ataques de contraseñas: su objetivo es descubrir contraseñas de acceso de usuarios para el acceso a determinados recursos. Se utilizan básicamente dos técnicas, por fuerza bruta o un ataque de diccionario.
- Ataques de recolección de información: ataque pasivo que consiste en recopilar información sensible de un recurso. Normalmente es preludio a un ataque posterior.

### *Segunda dimensión: objetivo del ataque*

Cubre el objetivo del ataque, es decir hacia donde está dirigido el vector de ataque anteriormente identificado. Al igual que la primera dimensión, se puede definir varios niveles con el fin de brindar un grado de clasificación más fino. Por ejemplo un ataque sobre un host, puede ser clasificado como un ataque sobre el S.O. Linux, distribución Ubuntu y finalmente versión 12.04. Ver Tabla 10.

| ID | Nivel 1 | ID | Nivel 2 | ID | Nivel3 | ID | Nivel 4 | ID | Nivel 5 |
|---|---|---|---|---|---|---|---|---|---|
| **O-1** | Hardware | 1 | Computador | 1 | Disco duro | | | | |
| | | | | 2 | Procesador | | | | |
| | | 2 | Equipo de red | 1 | Switch | | | | |
| | | | | 2 | Router | | | | |
| | | 3 | Periféricos | 1 | Teclado | | | | |
| **O-2** | Software | 1 | Sistema operativo | 1 | Windows | 1 | XP | | |
| | | | | | | 2 | Windows 7 | | |
| | | | | 2 | Linux | 1 | Ubuntu | 1 | 12.04.09 |
| | | | | | | | | 2 | 10.01 |
| | | | | | | 2 | CentOS | | |
| | | | | | | 3 | RedHat | | |
| | | 2 | Aplicación | 1 | Servidor | 1 | BDD | | |
| | | | | | | 2 | DNS | 1 | Bind |
| | | | | | | | | 2 | MS |
| | | | | 2 | Usuario | 1 | Skype | 1 | 4.x |
| **O-3** | Red | 1 | Protocolos | 1 | C.Transporte | 1 | TCP | | |
| | | | | | | 2 | UDP | | |
| | | | | 2 | C. Red | 1 | IP | | |
| | | | | 3 | C.Enlace | 1 | ARP | | |

**Tabla 10. Categorías de segunda dimensión de la taxonomía de ataques [46].**

### *Tercera dimensión: vulnerabilidades*

Cubre las vulnerabilidades que son explotadas sobre un objetivo identificado. A diferencia de la primera y segunda dimensión, no existe una clasificación estructurada para esta dimensión debido a la infinidad de vulnerabilidades existentes; en su lugar se emplea el identificador CVE (Common Vulnerabilities and Exposures) [48]. CVE es el nombre dado tanto al identificador de vulnerabilidad como al proyecto[3] que lo gestiona. Su objetivo es mantener un diccionario público de información acerca de las vulnerabilidades descubiertas; se aprovecha el hecho de que CVE se ha convertido en

[3] http://cve.mitre.org/

un estándar de facto para considerarlo como la tercera dimensión de esta taxonomía. La IETF define a una vulnerabilidad como una falla o debilidad en el diseño, implementación, u operación y mantenimiento de un sistema que puede ser explotada para violar la política de seguridad a la que se rige [49] . Por consiguiente se puede clasificar a las vulnerabilidades en tres grupos genéricos:

- Vulnerabilidades en el diseño: La falta de control en la definición de una contraseña y el uso de un algoritmo de cifrado débil son algunos ejemplos de vulnerabilidades de diseño. Aún cuando la implementación de un sistema sea segura, el origen y raíz de las vulnerabilidades están en el diseño mismo.
- Vulnerabilidades en la implementación: Dentro de este grupo se encuentran los errores en la validación de datos de entrada, violación de zonas de memoria y condiciones de carrera, por mencionar algunos.
- Vulnerabilidades en la operación y mantenimiento: Aún cuando un sistema haya sido diseñado e implementado de forma segura; si este no es desplegado y gestionado de manera adecuada se puede dar paso a ciertas vulnerabilidades. Desplegar un sistema de alta seguridad, perfectamente implementado, sobre un sistema operativo vulnerable, lo convierte también en un sistema vulnerable.

### *Cuarta dimensión: extensión de alcance*

Cubre la posibilidad de que un ataque que emplea un exploit, posea un payload cuya ejecución produzca un efecto de alcance más allá del esperado. Por ejemplo un gusano puede contener un payload de un troyano para eliminar algunos archivos (destrucción de información). El vector de ataque principal es el gusano; no obstante, una vez ejecutado el payload se eliminan algunos archivos, produciéndose un efecto mayor al que hubiese producido un gusano de forma individual. Cinco categorías pueden ser consideradas como punto de partida en esta dimensión:

| ID | Categoría |
|---|---|
| **E-1** | Vector de ataque de la primera categorías |
| **E-2** | Corrupción o destrucción de información |
| **E-3** | Divulgación de información |
| **E-4** | Robo de información |
| **E-5** | Subversión |

**Tabla 11. Categorías de la cuarta dimensión de una taxonomía de ataques.**

Finalmente mencionar que las dimensiones y categorías antes descritas no cubren todos los ataques existentes, su propósito es brindar un enfoque genérico que puede ser utilizado para clasificar un ataque. Se puede agregar nuevas dimensiones para clasificar a los ataques, en función del *impacto, costo o severidad* que causan por ejemplo.

De la misma manera, nuevas categorías pueden ser agregadas para brindar un grado de clasificación más fino; todo dependerá de las necesidades requeridas por el sistema o administrador que lo utiliza.

### 2.3.3 Acciones de respuesta por tipo de ataque

Un clasificador debe categorizar un ataque de la forma más precisa posible, con el fin de garantizar la ejecución de la acción de respuesta adecuada. En este sentido, la siguiente sección tiene por objetivo realizar la asignación de un grupo de acciones de respuesta, tomando como referencia las acciones definidas en la Tabla 100, a un ataque clasificado de acuerdo a la taxonomía descrita en la sección anterior.

Este apartado consta de dos tareas:

- La primera tarea consiste en elaborar una matriz, compuesta por las acciones de respuesta definidas en la Tabla 8 y los ataques pertenecientes a las categorías de nivel 1 definidas en la primera dimensión de la taxonomía de ataques.
- La segunda tarea consiste en elaborar una matriz, compuesta por las acciones de respuesta definidas en la Tabla 8 y los ataques clasificados considerando toda la taxonomía, es decir las cuatro dimensiones. El propósito es demostrar y hacer hincapié en la importancia de definir una taxonomía de ataques con un grado de finura adecuado, de tal forma que a cada tipo de ataque se asigne únicamente aquellas acciones que estén en capacidad de neutralizar dicho ataque. En esta sección se muestra únicamente un ejemplo de asignación dada la gran cantidad de escenarios posibles.

La matriz representada en la Tabla 12, hace una asignación de las acciones de respuesta que se pueden ejecutar cuando una intrusión pertenece a una de las nueve categorías de la primera dimensión de la taxonomía; no obstante, su grado de precisión es insuficiente y podría dar lugar a la ejecución de una acción de respuesta que no neutralice el ataque.

Considérese un escenario en donde una alerta ha sido clasificada como un ataque de ***Denegación de Servicio (A5)***; de acuerdo a la matriz construida, se pueden ejecutar 18 acciones de respuesta. Ahora, si se selecciona y se intenta ejecutar la respuesta R18, agregando una nueva política sobre el firewall de frontera, esta tendrá efecto solo si el ataque a más de ser clasificado como DoS, está basado en red. Si el ataque es de tipo DoS, pero está basado en host ya que se aprovecha de un bug en una aplicación para ocupar el 100% de CPU (consumo de recursos), entonces la acción de respuesta seleccionada es errónea y no neutralizará el ataque.

| Ataque / Respuesta | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 |
|---|---|---|---|---|---|---|---|---|---|
| R1 | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| R2 | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| R3 | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| R4 | √ | | √ | | | | | | |
| R5 | | | √ | | √ | √ | √ | √ | √ |
| R6 | √ | | √ | | | | | | |
| R7 | √ | | √ | | | | | | |
| R9 | √ | | √ | | | | | | |
| R10 | √ | √ | √ | √ | √ | √ | | √ | |
| R11 | √ | √ | √ | √ | √ | √ | | √ | |
| R12 | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| R13 | √ | √ | √ | √ | √ | √ | | √ | √ |
| R14 | √ | √ | √ | √ | √ | √ | | √ | √ |
| R15 | √ | √ | √ | √ | √ | √ | | √ | √ |
| R16 | √ | √ | √ | √ | √ | √ | | √ | √ |
| R17 | √ | √ | √ | √ | √ | √ | | √ | √ |
| R18 | | √ | √ | | √ | √ | | √ | √ |
| R19 | √ | | √ | √ | √ | √ | | √ | √ |
| R20 | | √ | √ | | √ | √ | | √ | √ |
| R21 | | √ | √ | | √ | √ | | √ | √ |
| R22 | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| R23 | | | √ | | √ | √ | | √ | √ |

**Tabla 12. Matriz que representa las acciones de respuesta que se pueden ejecutar en función del tipo de ataque al que pertenece una alerta de intrusión.**

Por consiguiente, es de suma importancia clasificar a un ataque de una forma precisa, estableciendo un grado de finura adecuado. Para ello se debe considerar la mayor cantidad de dimensiones y niveles posibles especificados por la taxonomía para clasificar un ataque.

En el escenario anteriormente planteado, el conjunto de acciones de respuesta que se podrían ejecutar ante un ataque de DoS (A5), basado en host (1), y además basado en consumo de recursos (1) se reducen a 12, tal como se observa en la Tabla 13:

| Ataque de DoS (A5) | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R 1 | R 2 | R 3 | R 4 | R 5 | R 6 | R 7 | R 8 | R 9 | R 10 | R 11 | R 12 | R 13 | R 14 | R 15 | R 16 | R 17 | R 18 | R 19 | R 20 | R 21 | R 22 | R 23 |
| √ | √ | √ | | √ | | | | | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| **Ataque de DoS (A5), basado en host (1) y basado en consumo de recursos (1) → A5-1-1** | | | | | | | | | | | | | | | | | | | | | | |
| √ | √ | √ | | | | | | | √ | √ | √ | √ | √ | √ | √ | | | √ | | | √ | |

**Tabla 13. Asignación de acciones de respuesta a un ataque de DoS, basado en host y en consumo de recursos.**

De esta manera, se reduce el conjunto de acciones de respuesta que puede ejecutar un IRS ante un ataque clasificado como A5-1-1; y se evita ejecutar una acción errónea que no neutraliza el ataque.

Ahora, hay que tener claro que el propósito de clasificar un ataque utilizando toda la taxonomía, no necesariamente es reducir el número de acciones de respuestas disponibles a ejecutar. Tal como se mencionó anteriormente, el propósito es seleccionar aquellas acciones de respuesta que puedan neutralizar un ataque, aún cuando ello implique aumentar el número de acciones de respuesta. Para demostrar lo antes mencionado, considérese un ataque con el gusano Blaster hacia una máquina con S.O. Windows XP.

Utilizando las cuatro dimensiones de la taxonomía, se clasifica al ataque Blaster de la siguiente manera:

| Ataque | 1° Dimensión | 2° Dimensión | 3° Dimensión | 4° Dimensión |
|---|---|---|---|---|
| **Blaster** | A2-2<br>Gusano (A2),<br>Propagación red (2) | O2-1-1-1<br>Software (O-2),<br>Sistema operativo (1),<br>Windows (1)<br>XP (1) | CAN-2003-0352 | DoS -> Inundación TCP |
| | A5-2-2<br>DoS(A5),<br>Basado en red (2)<br>Inundación TCP (2) | O3-1-1-1<br>Red (O3)<br>Protocolos (1)<br>C. Transporte (1)<br>TCP (1) | | |

**Tabla 14. Clasificación de ataque Blaster utilizando la taxonomía de Hansman.**

Como se observa en la Tabla 14, el vector de ataque principal de Blaster es un Gusano (A2) que se propaga a través de la red; no obstante, de acuerdo a la cuarta dimensión, el gusano causa un efecto mayor al de una simple propagación en la red. Se produce la denegación de servicio por inundación de paquetes TCP, por lo que el momento de asignar las acciones de respuesta a ejecutar ante un ataque Blaster, se deben considerar los dos vectores de ataque A2-2-O2-1-1-1 y A5-2-2-O3-1-1-1.

El grupo de acciones de respuesta a asignar al ataque Blaster se muestra en la Tabla 15. Observe que si solo se considera el primer vector de ataque el número de acciones

de respuesta es de 15, mientras que si se consideran ambos vectores, el número de acciones se incrementa a 18.

| Ataque Blaster (A2-2-O2-1-1-1) | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R 1 | R 2 | R 3 | R 4 | R 5 | R 6 | R 7 | R 8 | R 9 | R 10 | R 11 | R 12 | R 13 | R 14 | R 15 | R 16 | R 17 | R 18 | R 19 | R 20 | R 21 | R 22 | R 23 |
| √ | √ | √ | | | | | | | √ | √ | √ | √ | √ | √ | √ | √ | √ | | √ | √ | √ | |
| **Ataque DoS (A5-2-2-O3-1-1-1)** | | | | | | | | | | | | | | | | | | | | | | |
| √ | √ | √ | | √ | | | | | | | √ | √ | | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| **Ataque Blaster y DoS** | | | | | | | | | | | | | | | | | | | | | | |
| √ | √ | √ | | √ | | | | | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |

Tabla 15. Asignación de acciones de respuesta a un ataque Blaster.

## 2.4 Sistema autónomo de respuesta a intrusiones basado en ontologías

Cómo parte de un proyecto de investigación emprendido por la Universidad Politécnica de Madrid[4], se ha desarrollado una propuesta de arquitectura e implementación de un prototipo de un sistema autónomo de respuesta a intrusiones basado en ontologías. Dicho IRS, es un sistema de respuesta a intrusiones, cuyo principal aporte consiste en la adición de la coherencia semántica al conjunto de características que se espera cumpla un IRS. Aquello es imprescindible, ya que en un entorno de red heterogéneo, donde intervienen diferentes IDSs, cada uno maneja alertas en diferentes formatos y diferente sintaxis [10], [11]. Cómo es de esperar, el IRS basado en ontologías debe cumplir con ciertas características que definen a un IRS en general, como por ejemplo tener la capacidad de seleccionar una respuesta óptima de entre un conjunto de respuestas recomendadas, en función del costo que implica ejecutarlo. Dichas características ya fueron abordadas en la sección 2.2; no obstante, la presente sección describe el AIRS basado en ontologías desde un punto de vista práctico. Esto servirá como preámbulo para la siguiente sección, donde se determinan las bases para la propuesta de la arquitectura del *Módulo de ejecución de respuestas.*

### 2.4.1 Arquitectura

En la Figura 5, se presenta la arquitectura del AIRS basado en ontologías. El objetivo del AIRS es escoger la respuesta óptima de entre un conjunto de respuestas recomendadas; para ello hace uso de una ontología que es representada mediante un lenguaje formal de representación de conocimiento. Las clases que conforman la ontología son utilizadas para representar ciertas políticas que son definidas por el administrador, y en base a los cuales se realiza el proceso de inferencia. A continuación se describe cada uno de los módulos que conforman la arquitectura.

[4] Proyecto CENIT Segur@, abalado por el Centro para el Desarrollo de Tecnología Industrial, España.

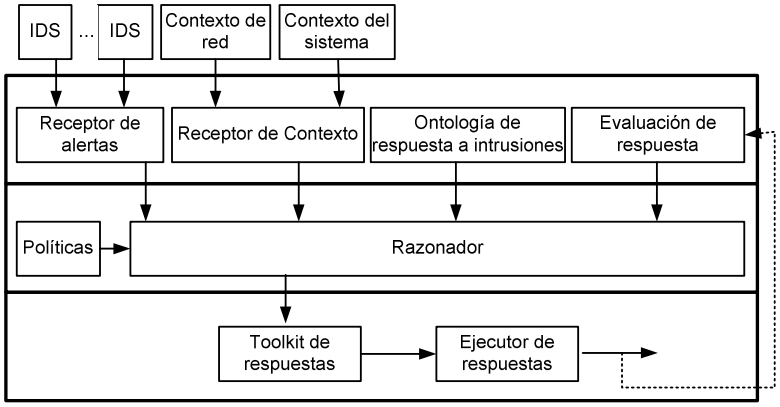



**Figura 5. Arquitectura del Sistema de Respuesta Automática a Intrusiones basado en ontologías [10].**

*Receptor de alertas:* Recepta las alertas de intrusiones provenientes de diferentes IDSs con diferentes formatos y sintaxis, y posteriormente mapea los campos incluidos en estas alertas a sus conceptos equivalentes definidos en la ontología.

*Contexto de red:* El objetivo es calcular un *parámetro de anomalía de red,* en base a la diferencia existente entre dos snapshots del tráfico de red requerido: el primero, que es obtenido en condiciones normales de operación, y; el segundo, que es obtenido en tiempo real cuando una intrusión es detectada.

*Contexto del sistema:* El objetivo es calcular un *parámetro de anomalía del sistema,* en base a la diferencia entre dos snapshots que evalúan ciertos parámetros del sistema antes y después del ataque: el número de procesos activos, número de procesos zombies, porcentaje de uso de memoria y espacio en disco son algunos ejemplos de ellos.

*Receptor de contexto:* Recepta la información de contexto de red y del sistema y los mapea a sus correspondientes conceptos definidos en la ontología.

*Ontología de respuesta a intrusiones:* Define formalmente toda la información del *dominio de respuestas a intrusiones* necesaria para inferir la respuesta óptima ante una intrusión. Dicho dominio, como se puede constatar en la Figura 39, está conformado por todas las entidades necesarias que definen el entorno del problema en cuestión: la red propiamente dicha, componentes del sistema, sistemas de detección de intrusiones, sistemas de respuesta a intrusiones, alertas de intrusiones, contexto de red, contexto del sistema, respuestas y resultado de una respuesta. Cada una de estas entidades es representada mediante clases, propiedades y sus relaciones, utilizando el lenguaje estándar OWL (Ontology Web Language) definida por la W3C [50]. En la Figura 6 se puede observar la ontología completa de respuesta a intrusiones.

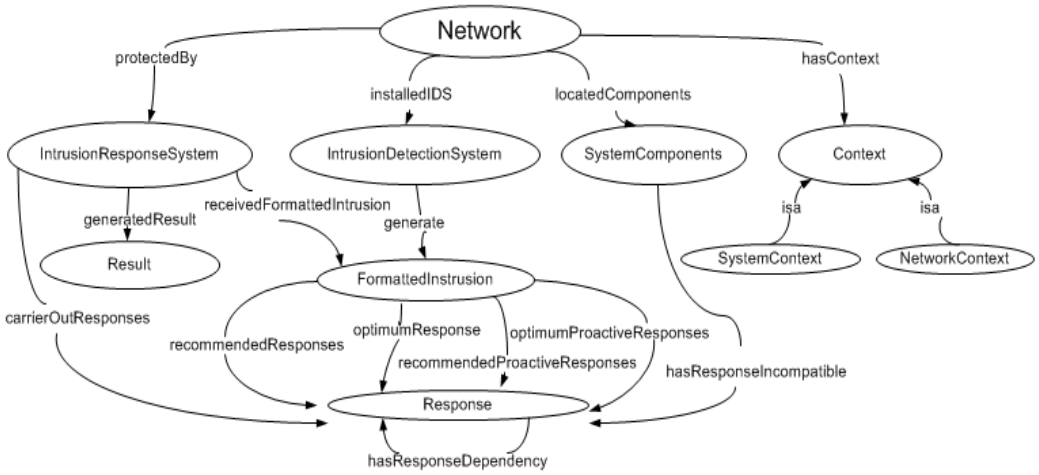



**Figura 6. Ontología de respuesta a intrusiones [10].**

*Políticas:* Están compuestas por un conjunto de reglas que definen el comportamiento del AIRS. Estas reglas son definidas por el administrador del sistema utilizando el lenguaje SWRL (Semantic Web Rules Language) [51] y conforman las denominadas *métricas de respuesta.* Estas métricas son abordadas en la sección 2.4.3.

*Razonador:* Es el componente principal del AIRS y es el encargado de inferir la respuesta óptima para una intrusión dada. Para ello toma como entrada las *políticas* definidas por el administrador y una *instancia de la ontología* antes mencionada. En este trabajo se emplea el motor de inferencia Bossam, un razonador semántico con soporte nativo para ontologías OWL y SWRL[52].

*Toolkit de respuestas:* Es el conjunto de *acciones de respuesta* disponibles que pueden ser inferidas por el AIRS; dichas respuestas pueden ser activas y pasivas.

*Ejecutor de respuestas:* Lo constituyen los componentes de seguridad que intervienen en la ejecución de una acción de respuesta.

### 2.4.2 Proceso de inferencia

Como se mencionó anteriormente el objetivo del AIRS basado en ontologías es inferir la respuesta óptima ante la detección de una intrusión. Entonces, cada vez que se recibe una alerta de intrusión se lleva a cabo un proceso de inferencia, cuyo diagrama de flujo se muestra en la Figura 23, y que esencialmente consta de tres fases: *recolección de información, inferencia de respuestas recomendadas e inferencia de la respuesta óptima*. Ver Figura 7.

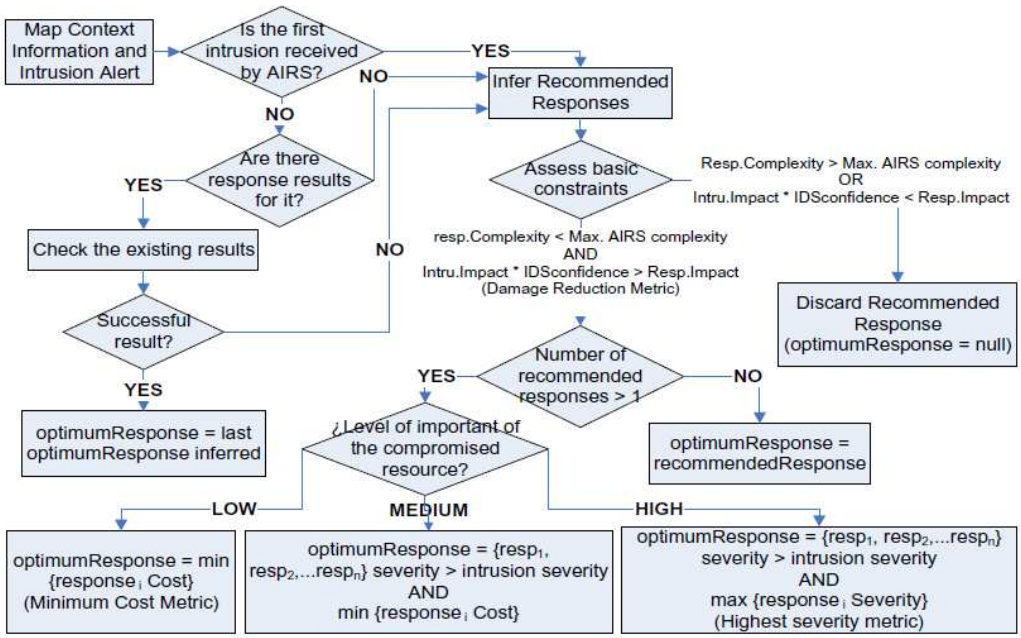



Figura 7. Proceso de inferencia de la respuesta óptima [11].

i. **Recolección de información:** Se obtiene la información de contexto, a través de los módulos de contexto de red y contexto del sistema, y las alertas desde los diferentes IDSs. Posteriormente, esta información se mapea a los conceptos equivalentes de la ontología utilizando el lenguaje OWL, para ello se usan los receptores de contexto y alertas respectivamente.

ii. **Inferencia de respuestas recomendadas:** En primera instancia mencionar que el AIRS basado en ontologías considera que dos ataques son **similares** cuando son del mismo tipo, afectan al mismo recurso y ocasionan los mismos cambios de contexto. Ahora bien, en esta segunda fase de éste proceso de inferencia el sistema verifica si ha recibido una alerta de intrusión de similares características con anterioridad:

   a. Si no se ha recibido alertas similares implica que no existen resultados de respuestas previas; por consiguiente, el AIRS infiere un conjunto de respuestas recomendadas, en función de la información obtenida en la fase 1 y aplicando las *políticas* establecidas por el administrador.

   b. Si previamente ya se han recibido alertas de intrusiones similares, el sistema verifica si hay resultados de respuestas anteriores utilizadas para neutralizar dichas intrusiones: si existen respuestas satisfactorias, entonces se elegirá la última respuesta ejecutada como óptima; caso contrario, se inferirá un conjunto de respuestas recomendadas, en función de la información obtenida en la fase 1 y aplicando las *políticas* establecidas por el administrador.

iii. **Inferencia de la respuesta óptima:** En función de la importancia del recurso afectado y de los parámetros usados en las *métricas de respuesta*, el AIRS infiere la respuesta óptima de entre las respuestas recomendadas. Los parámetros usados en las métricas son: *impacto de intrusión, severidad de la respuesta, impacto de la respuesta, confianza del IDS, nivel de importancia del recurso afectado y eficiencia de la respuesta*. Estos parámetros son usados para definir 3 métricas de respuesta del AIRS: *métrica de reducción de daño, métrica del mínimo costo y métrica de más alta severidad* que son descritos en la sección 2.4.3.

a. **Impacto de la intrusión (iI):** Costo del daño causado por un ataque que se ha materializado. El valor para éste parámetro es asignado por el Receptor de Alertas, cuyo cálculo depende de la importancia del recurso atacado y de la severidad de la intrusión.
b. **Impacto de la respuesta (iR):** Costo del daño causado al ejecutar una respuesta. El valor para éste parámetro dependerá entre otras cosas, del número de recursos afectados por el despliegue de la respuesta, importancia de los recursos y del nivel de degradación de los recursos.
c. **Confianza del IDS (Co):** Determina la confianza de que un IDS genere una alerta de intrusión real. El valor de éste parámetro está determinado por el número total de intrusiones reales y el número de falsos positivos y negativos generados.
d. **Eficiencia de una respuesta (eR):** Determina el nivel de éxito de una respuesta en contra de una intrusión en particular. El valor de éste parámetro está determinado por el promedio de los factores *de eficiencia parcial de respuesta*, que son calculados después de la ejecución de cada respuesta en base a la información de contexto de red y sistema.
e. **Severidad de la intrusión (sI):** Determina cuan negativo es el efecto causado por una intrusión en la operación normal de la red, al comprometer a un recurso objetivo. El valor de éste parámetro es definido por el IDS y depende del tipo de intrusión.
f. **Severidad de la respuesta (sR):** Determina cuan negativo es el efecto causado en la operación normal de la red, al ejecutar respuesta para contrarrestar una intrusión.

### 2.4.3 Definición de las métricas de respuesta

#### Métrica de reducción de daño

El AIRS aplica esta métrica independientemente de la importancia del recurso y su objetivo es lograr un equilibrio entre el costo de ejecutar una respuesta y el costo de que un ataque se materialice. El costo de la respuesta y ataque son cuantificados por medios del impacto respectivo que fueron definidos en la sección anterior. La métrica satisface la siguiente ecuación:

$$iI * Co > iR$$

$donde$:
$iI = impacto\ de\ la\ intrusión$
$Co = confianza\ del\ IDS$
$iR = impacto\ de\ la\ respuesta$

De éste modo todas las *respuestas recomendadas* son evaluadas; filtrando aquellas respuestas cuyo impacto es mayor al impacto que causaría la intrusión.

#### Métrica del mínimo costo

El AIRS aplica esta métrica solo si los recursos afectados no tienen gran importancia dentro de la organización y su objetivo es contrarrestar una intrusión con la respuesta de menor complejidad y severidad, de tal forma que se **minimice** el *Costo Total de Respuesta (ctR)*. La métrica satisface las siguientes ecuaciones:

$$ctR = iR + cdR$$

$$Min\{ctR\}$$

$donde$:
$ctR = costo\ total\ de\ respuesta$
$iR = impacto\ de\ la\ respuesta$
$cdR = costo\ de\ despliegue\ de\ respuesta$

Donde el cdR (Costo de Despliegue de Respuesta) representa el costo en el que incurre una organización ante el despliegue de una respuesta, en términos de recursos requeridos para llevarlo a cabo (por ejemplo número de dispositivos requeridos, servicios involucrados, archivos a respaldar, etc.). Finalmente mencionar que si varias respuestas tienen el mismo ctR, se escoge la de menor valor, es decir el de menor complejidad.

### Métrica de alta severidad y eficiencia

El AIRS aplica esta métrica si los recursos afectados son críticos y de gran importancia dentro de la organización y su objetivo es contrarrestar una intrusión con la respuesta más severa y eficiente (en base a respuestas exitosas previas), independiente de su costo y complejidad. La métrica debe cumplir con las siguientes ecuaciones:

$$eR * sR \geq sI$$

$$Max\{eR * sR\}$$

$donde$:
$eR = eficiencia\ de\ respuesta$
$sR = severidad\ de\ la\ respuesta$
$sI = severidad\ de\ la\ intrusión$

De acuerdo a las ecuaciones antes descritas: la primera, permite seleccionar todas las respuestas cuya eficiencia y severidad es más alta que la severidad de la intrusión; la segunda, garantiza que la respuesta a ejecutarse es la de más alta severidad y cuyo éxito (eficiencia) en ejecuciones anteriores contra intrusiones similares también es relativamente alta.

# 3 Planteamiento del problema

Como se puede observar en la Figura 8, el AIRS basado en ontologías propone una arquitectura modular en donde se puede identificar cuatro módulos bien diferenciados: módulo de recopilación de información, módulo de parseo de información, módulo de razonamiento y módulo de ejecución de respuestas. Los tres primeros módulos mencionados han sido abordados casi en su totalidad en el proyecto Segur@; no obstante, el nivel de ejecución de respuestas presenta ciertas limitaciones que el presente trabajo intenta solventarlas. En virtud de ello, esta sección se enfoca exclusivamente en el nivel de ejecución de respuestas y tiene por objetivo establecer un marco inicial, a partir del cual se realizará una propuesta de arquitectura que permita la ejecución de acciones de respuesta del AIRS.

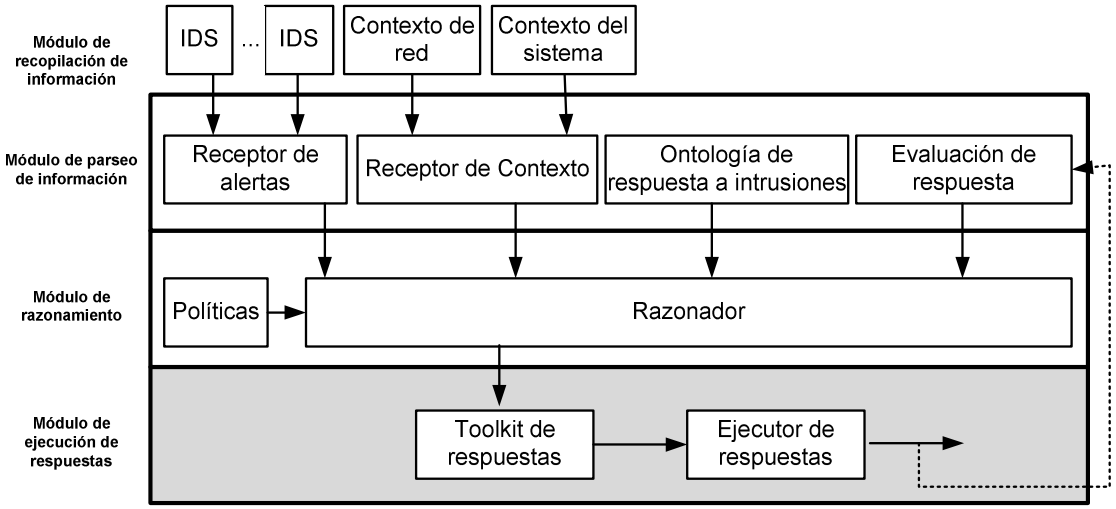



Figura 8. Módulos de la arquitectura del AIRS basado en ontologías.

## 3.1 Estado actual del módulo de ejecución de respuestas

### 3.1.1 Requisitos establecidos inicialmente

El AIRS basado en ontologías establece ciertos requisitos funcionales y de salida para el módulo de ejecución de respuestas, que se muestran en la Tabla 16:

| **Requisitos funcionales del AIRS** | |
|---|---|
| **REQF01** | El sistema responderá a ataques de red |
| **REQF02** | El sistema responderá a ataques de host |
| **Requisitos de salida del AIRS** | |
| **REQS01** | El sistema podrá generar múltiples tipos de respuesta en base a un catálogo de acciones de respuesta |
| **REQS02** | Las acciones de respuestas serán reactivas (activas o pasivas) o proactivas. |

Tabla 16. Requisitos funcionales y de salida del AIRS que fueron establecidos inicialmente.

El REQF01 y el REQF02 hacen referencia a la necesidad de que el AIRS basado en ontologías debe estar en la capacidad de neutralizar intrusiones detectadas tanto por un NIDS como por un HIDS; consecuentemente, debe estar en la capacidad de interactuar con múltiples componentes de seguridad locales y remotos para poder neutralizar las intrusiones.

El REQS01 y el REQS02 hacen referencia a la necesidad de que el AIRS basado en ontologías cuente con un conjunto de acciones de respuesta sobre los cuales se inferirá la respuesta óptima. Dichas acciones deben ser: proactivas, por consiguiente deben ser ejecutadas sobre dispositivos que se encuentran en línea con la potencial intrusión, y: reactivas, pudiendo ser a su vez pasivas o activas. Estas respuestas fueron abordadas en la sección 2.2 y 2.3.

El AIRS basado en ontologías, de momento, cuenta con un catálogo de respuestas que contiene 3 acciones:

- Envió de correo electrónico: Una respuesta pasiva que envía una notificación al administrador de la ocurrencia de un ataque.
- Bloqueo de puertos: Una respuesta activa de protección que añade una regla al firewall de frontera con el objetivo de bloquear aquellos paquetes dirigidos a un puerto en particular.
- Despliegue de una red señuelo: Una respuesta activa de decepción que cumple con dos tareas: primero, mitiga el ataque redirigiéndolo hacia otro sitio; y segundo, analiza al atacante y su modo de actuación.

Las tres respuestas antes mencionadas, son ejecutadas de forma local, es decir sobre el mismo host en el que se encuentra instalado el AIRS basado en ontologías.

### 3.1.2 Componentes que intervienen

Los componentes de la arquitectura del AIRS que intervienen en la ejecución de una de respuesta se muestran en la Figura 9.

El *catálogo de acciones* contiene el conjunto de acciones de respuesta que están disponibles para ejecutar. Cada acción de respuesta debe ser agregado al catálogo; para ello, se debe crear una nueva instancia de la clase *Response* definida en la ontología del AIRS (ver Tabla 17), que básicamente define las condiciones bajo las cuales se ejecutará dicha respuesta.

Una de las acciones de respuesta, contenidas en dicho catálogo, será inferida por el razonador para ser ejecutada. El razonador lanza la ejecución de una respuesta invocando su nombre, *responseAction* de la Tabla 17, y pasa los *parámetros necesarios* hacia el *toolkit de respuestas* para ejecutarla.

| | |
|---|---|
| orientedToIntrusionType | Hacia qué tipo de intrusión va dirigida la respuesta. Los tipos son definidos en la taxonomía, descrita en la sección 2.3.2 |
| orientedToIntrusionLocation | Desde donde tuvo lugar el ataque: desde la red local o externa. |
| orientedToAttackerType | Hace referencia al tipo de atacante, que dependerá del nivel de conocimiento acerca del ataque ejecutado. |
| orientedToIntrusionConsequence | Consecuencia que tiene el ataque en la red. |
| orientedToIntrusionSourceType | El tipo de recurso que fue utilizado para ejecutar el ataque. |
| orientedToIntrusionTargetType | El tipo de recurso en la red que es atacado. |
| orientedToIntrusionDetectionTime | Momento en que fue *detectado* el ataque: antes, durante o despúes de que se efectuó el ataque propiamente dicho. |
| responseAction | Nombre de la acción de respuesta a ejecutar. |
| responseComplexity | Complejidad de la respuesta*. |
| responseSeverity | Severidad de la respuesta*. |
| responseCost | Costo de la respuesta*. |
| responseType | Tipo de respuesta: pasiva o activa (reactiva, proactiva). |
| responseImpact | Impacto de la respuesta. |

**Tabla 17. Parámetros de la clase response de la ontología del AIRS. (*) Estos parámetros fueron definidos en la sección 2.4.2.**

El toolkit de respuestas lanza una orden sobre el *ejecutor de respuestas*, que es el encargado de poner en marcha mecanismos de respuesta reales para neutralizar el ataque.

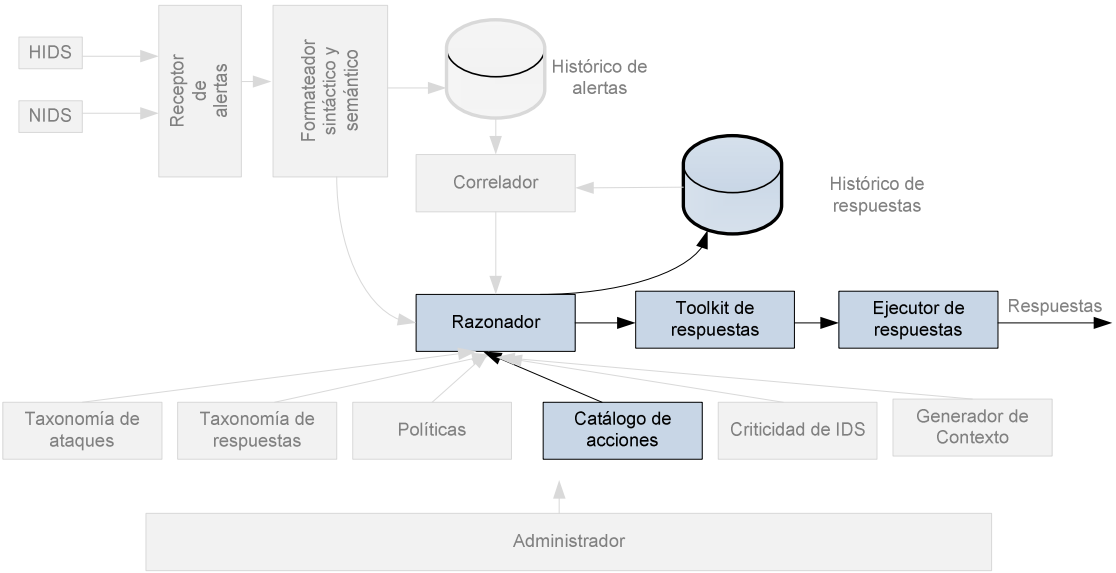



**Figura 9. Componentes que intervienen en la ejecución de respuestas.**

Finalmente se mantiene un histórico de respuestas que almacena toda y cada una de las respuestas ejecutadas. Para cada una se indica el tipo de ataque al que responde y el éxito o no de la ejecución.

### 3.1.3 Desarrollo actual

Para la implementación del AIRS se ha empleado el lenguaje de programación Java, cuyo diagrama de clases se presenta en la Figura 10. Se resalta las clases que intervienen en la ejecución de respuestas.

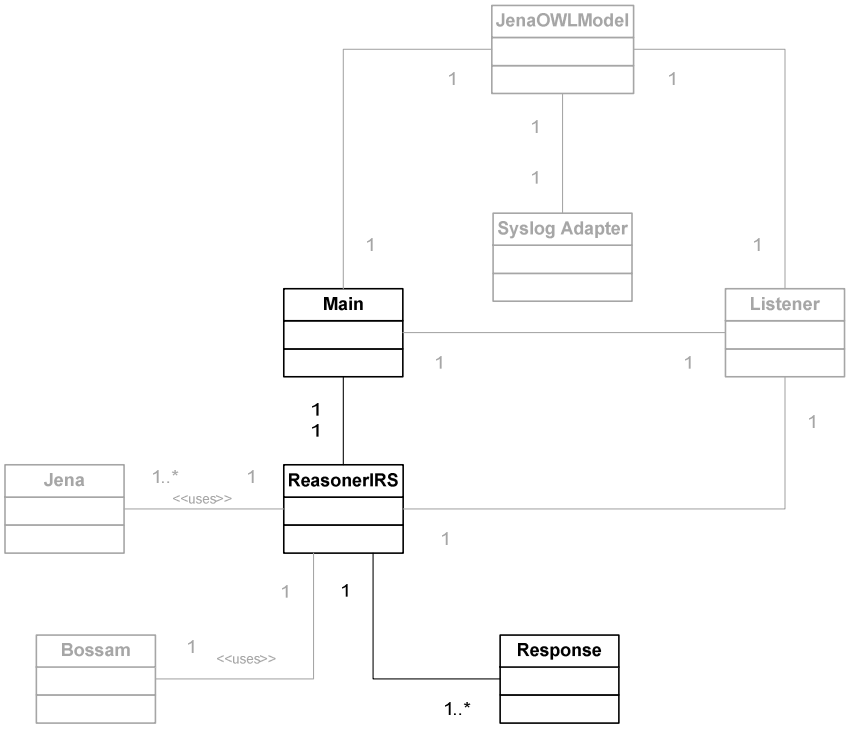



Figura 10. Diagrama de clases del AIRS basado en ontologías.

Las clases que intervienen en el proceso de ejecución de respuestas son la clases: Main, ReasonerIRS y Response.

La clase *main* es la clase principal del razonador del AIRS. Cada vez que se detecta una intrusión, la clase main invoca al *ReasonerIRS*, que representa al razonador del AIRS, y se encarga de inferir las respuestas recomendadas y posteriormente la respuesta óptima. Esta clase es la encargada de llamar a una instancia de la clase *response*, que representa a la acción de respuesta inferida por el AIRS, proporcionándole los parámetros necesarios para su ejecución.

Además se debe tener presente las siguientes consideraciones:

- Cada acción de respuesta se debe implementar mediante una clase desarrollada en Java.
- Cada clase que implementa una respuesta que debe definir los parámetros necesarios para su ejecución. Por ejemplo para la acción de respuesta que despliega la red señuelo se crea la clase *ThrowHoneyNet*, que requiere como parámetros de entrada: *IP Origen, IP Destino, Puerto y Protocolo*.

- Cuando una respuesta es inferida el *ReasonerAIRS* invoca una instancia de *ThrowNoheyNet* enviando los parámetros antes mencionados para su ejecución; dichos parámetros son obtenidos de la alerta de intrusión que origina la respuesta.
- Cada clase implementa la lógica de la respuesta de forma independiente, el AIRS no necesita conocer los detalles de dicha implementación.
- Para que una respuesta pueda ser ejecutada por el ReasonerAIRS, el nombre de la clase que implementa la respuesta debe coincidir con el valor que contiene la propiedad *responseAction* de la ontología.

## 3.2 Problemas del módulo de ejecución de acciones de respuestas

De acuerdo a los requisitos inicialmente planteados para el desarrollo del AIRS basado en ontologías y que fueron descritos en la Tabla 16, el AIRS basado en ontologías luego de inferir la respuesta óptima, tiene que estar en la capacidad de ejecutar acciones de respuesta sobre diferentes componentes de seguridad, dado que recibe alertas de intrusiones tanto de NIDS como de HIDS. Es decir, debe ser posible ejecutar tanto acciones de respuesta basadas en red como acciones de respuesta basadas en hosts (Ver Tabla 8).

Además el AIRS basado en ontologías, debe contar con un catálogo de acciones de respuesta de entre las cuales será seleccionada la repuesta óptima. Dichas respuestas deben ser activas y pasivas (Ver Tabla 8).

Sin embargo, se ha podido identificar tres problemas de consideración en el módulo de ejecución de respuestas del AIRS basado en ontologías:

1. **Únicamente permite la ejecución local de acciones de respuesta:** El módulo de ejecución de respuestas del AIRS actualmente puede ejecutar acciones de forma local; es decir en la misma máquina sobre la que se ejecuta el AIRS. No obstante, como se menciona en el capítulo 2, una de las características fundamentales de un IRS autónomo es su capacidad para ejecutar acciones de respuestas interactuando con otros componentes de seguridad, como por ejemplo:
    a. Firewalls de frontera: para agregar reglas de filtrado que bloqueen el flujo de tráfico de un atacante que está localizado en la red externa.
    b. Firewalls personales: para agregar reglas de filtrado que bloqueen el flujo de tráfico de un atacante que está localizado en la red interna.
    c. Routers de frontera o internos: para agregar listas de control de acceso para bloquear determinado flujo de tráfico.

d. Hosts: para terminar un proceso o servicio, para deshabilitar un usuario, para terminar conexiones, para respaldar un archivo, entre otros.

Además, el módulo de ejecución de respuestas no define función alguna para la implementación de las acciones de respuesta. El ReasonerAIRS simplemente llama a una respuesta por su nombre y envía los parámetros requeridos; sin embargo no se preocupa de los detalles de implementación: ¿la ejecución es local o remota?, en caso de ser remota ¿sobre qué hosts ejecutarse?, ¿son requeridos parámetros adicionales a los que se disponen en las alertas de intrusión?, etc.

2. **Escalabilidad limitada:** Ante la carencia de una arquitectura distribuida del módulo de ejecución de respuestas, agregar nuevas acciones de respuesta que se ejecuten sobre diferentes componente se seguridad puede ser una tarea muy compleja; la arquitectura propuesta para el AIRS basado en ontologías no provee funciones que faciliten el despliegue de las acciones de respuesta ya que la lógica de la respuesta le es indiferente. No obstante, hay determinadas funciones como: el establecimiento de conexión con un componente de seguridad remoto, cifrado y descifrado de la solicitud de una acción de respuesta y otras funciones de control; que son requeridas por todas las acciones de respuesta, independiente de su lógica, y que podrían ser provistas por un marco común de comunicación.
3. **El catálogo de acciones de respuesta es muy reducido:** El AIRS basado en ontologías, al momento cuenta con un catálogo de tres acciones de respuesta, como se mencionó en la sección 3.1.1: envío de correo electrónico, despliegue de una red señuelo y bloqueo de puertos. Sin embargo, es necesario contar con un conjunto más amplio de acciones que permitan validar el AIRS basado en ontologías.

Los problemas anteriormente planteados se abordan en dos tareas separadas:

1. **Propuesta de arquitectura de un módulo de ejecución de respuestas distribuido:**
   Para resolver los problemas 1 y 2 se propone una arquitectura de un *módulo ejecutor de respuestas distribuido*. Dicha arquitectura suplanta el componente ejecutor de respuestas original que se muestra en la Figura 11.

La forma en que el Razonador (ReasonerAIRS) llama a las acciones de respuesta se mantiene, pues uno de los objetivos es evitar cambio alguno en la arquitectura del AIRS basado en ontologías. La especificación de requisitos, el diseño e implementación se abordan en los capítulos 4, 5 y 6.

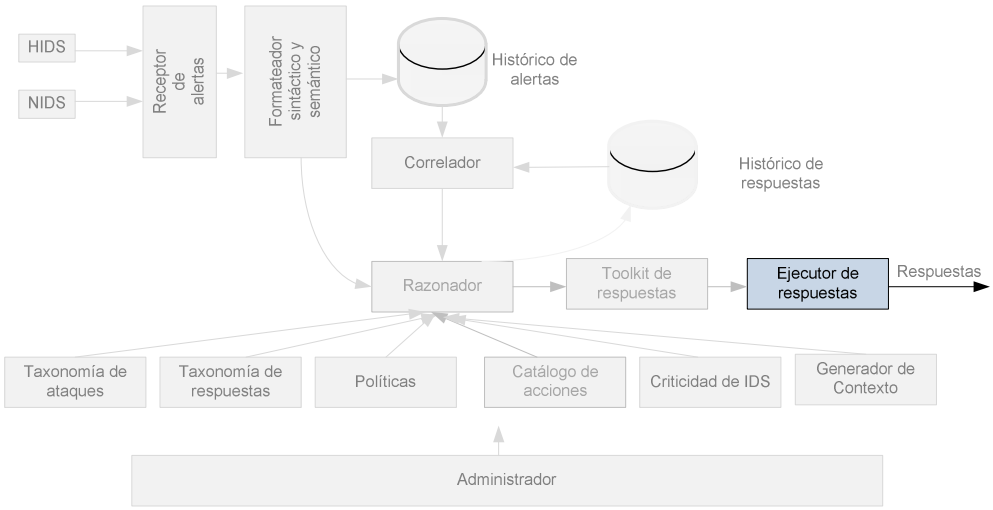



Figura 11. Módulo ejecutor de respuestas del AIRS basado en ontologías.

2. **Dotación de *pruebas de concepto* de acciones de respuesta que permitan una futura validación del AIRS basado en ontologías:**

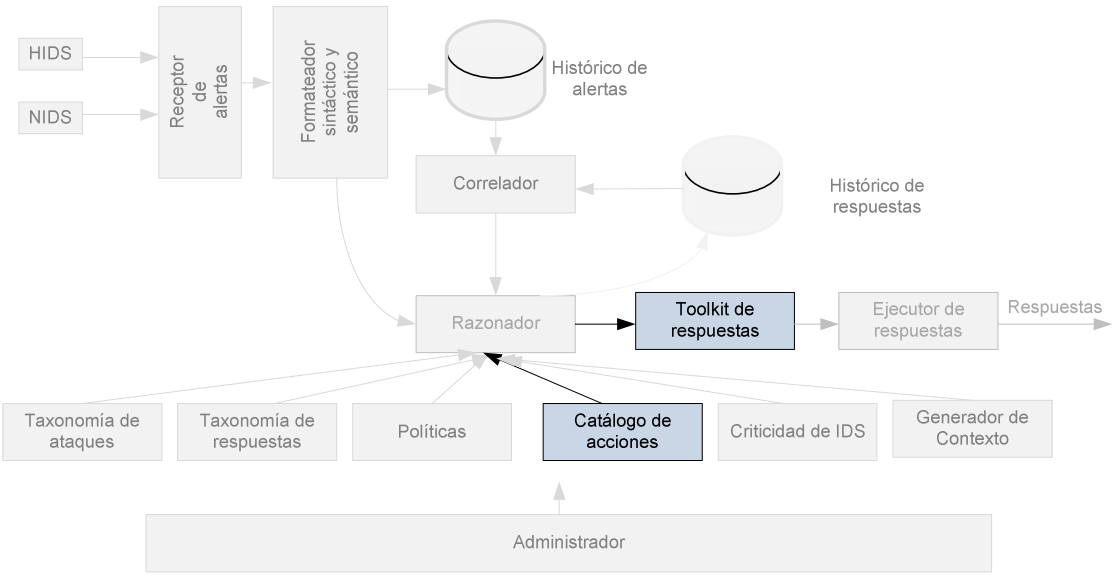



Figura 12. Submódulos para adición de nuevas acciones de respuesta.

La segunda tarea consiste en dotar de nuevas acciones de respuesta que serán implementadas a modo de pruebas de concepto; dichas pruebas de concepto servirán para validar el módulo ejecutor de respuestas distribuido y en futuro permitirá la validación del AIRS basado en ontologías. Los aspectos relevantes de implementación de describen en el capítulo 6.

# 4 Especificación de requerimientos.

El presente capítulo tiene por objetivo definir los requerimientos del módulo de ejecución de respuestas. Para ello se sigue el siguiente procedimiento:

1. Se obtienen los requerimientos generales que intentan resolver los problemas 1 y 2 descritos en el capítulo 3; dichos requerimientos son representados mediante un diagrama de casos de uso y posteriormente se describe el fundamento que justifica su inclusión.
2. En teoría, los requerimientos del sistema simplemente deben describir el comportamiento externo del sistema y sus restricciones operativas. Es decir que no deben especificar cómo se debe diseñar e implementar el sistema. No obstante, como señala Sommervile en [53], en la práctica aquello no siempre es posible de cumplir, ya que en ocasiones es necesario plantear un bosquejo inicial de arquitectura que ayude a estructurar la especificación de requerimientos. En virtud de ello, en esta segunda parte se plantea un bosquejo con ciertos componentes, que en principio, formarán parte de la arquitectura propuesta. Existen dos motivaciones para ello:
    a. Organizar los requerimientos conforme a ciertos submódulos que conforman el ejecutor de respuestas distribuido, y;
    b. El módulo ejecutor de respuestas distribuido debe interoperar con otro sistema ya existente, el AIRS basado en ontologías (razonador e IDSs); que impone ciertos requerimientos en el ejecutor de respuestas.
3. Se realiza la especificación de requerimientos funcionales y no funcionales de cada uno de los submódulos que conforman el ejecutor de respuestas distribuido.

## 4.1 Casos de uso

En la Figura 13 se muestra un diagrama de casos, cuyo objetivo es obtener los requerimientos generales que debe cumplir el ejecutor de respuestas. Se han incluido cuatro actores: *IDS, razonador, ejecutor de respuestas y componente de seguridad*. El IDS, el razonador y el componente de seguridad son sistemas existentes, es decir ya han sido implementados; no obstante, se incluyen ya que imponen ciertos requerimientos a tomar en cuenta en el diseño del ejecutor de respuestas.

- **IDS:** es un actor principal que representa a un sistema de detección de intrusiones. Su función es: monitorizar una fuente de datos que puede ser el tráfico de red, archivos de logs, llamadas al sistema, etc. (CU01), y; disparar una alerta de intrusión para informar al *razonador* (CU02). Como axioma de lo antes mencionado, el IDS puede ser de cualquier tipo: basado en host, basado en red, basado en aplicación o basado en nodo de red;

consecuentemente debe estar preparado para responder ante diferentes tipos de alertas.

- **Razonador:** es un actor principal que representa al *ReasonerAIRS*. Su función, como ya se ha descrito en secciones anteriores, es inferir la respuesta óptima (CU03) de un catálogo de acciones de respuesta. Posteriormente invoca la ejecución de la respuesta seleccionada (CU04), otorgándole "algunos" parámetros necesarios que son obtenidos de la alerta de intrusión.
- **Ejecutor de respuestas:** Es un actor principal que se encarga de ejecutar una acción de respuesta sobre un componente de seguridad que está localizado local o remotamente. Puesto que se enfoca en este punto, cada caso de uso es descrito a continuación, fundamentando su inclusión dentro del sistema.
- **Componente de seguridad:** Lleva a cabo la ejecución real de una acción de respuesta utilizando los comandos propios del dispositivo (CU09). Un firewall, router, tcpwrapper, gestor de usuarios, gestor de procesos, entre otros son algunos ejemplos de componentes de seguridad.

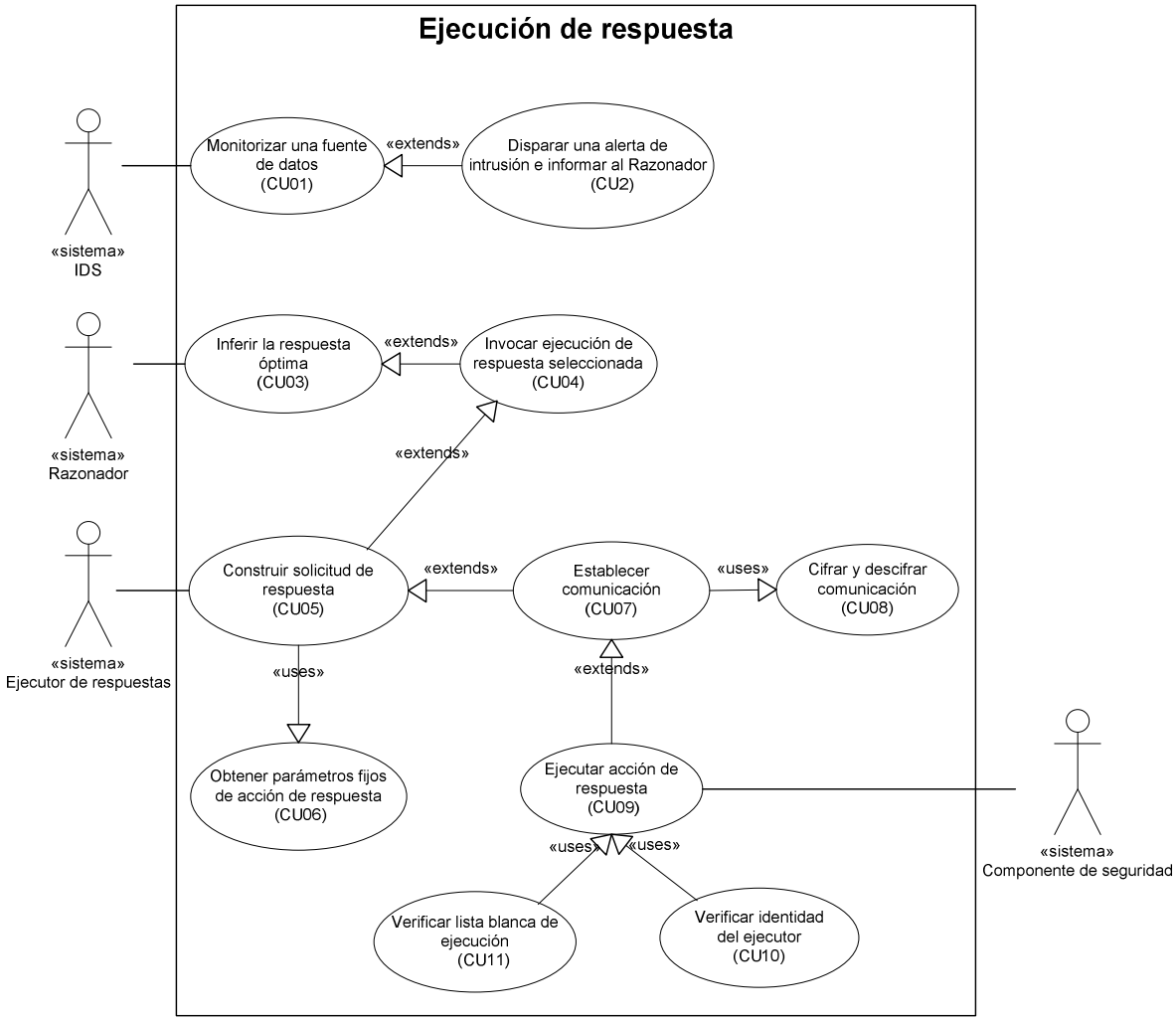



**Figura 13. Diagrama de casos de uso del módulo de ejecución de respuestas.**

### 4.1.1 Justificación

**CU05. Construir solicitud de respuesta.**

**Fundamento:** Una vez que el razonador selecciona e invoca la respuesta óptima se requiere la ejecución de una acción de respuesta sobre un componente de seguridad. En un principio, era suficiente contar con firewalls o routers de frontera, que se encargaban de agregar políticas o listas de control de acceso respectivamente [54]. Sin embargo, estos dispositivos no podían garantizar un control de acceso en contra de ciertos ataques que utilizan comunicaciones cifradas y más aún de aquellos que se originan al interior de la red. Por consiguiente, es preciso contar con un sistema que sea capaz de interactuar con componentes de seguridad diversos. En este caso es necesario ***construir una solicitud de acción de respuesta*** que contenga los ***parámetros necesarios*** para su ejecución en determinado componente.

*Algunos* de los parámetros necesarios para construir la solicitud son proporcionados por el *razonador*; quien a su vez los obtiene de las alertas de intrusiones. Dichos parámetros deben ser recibidos utilizando algún método. Además, es importante mencionar que las alertas no siempre contienen la misma información, tal como se describe en la sección 2.1.2, dependerá del tipo de IDS.

**CU06. Obtener parámetros fijos de acción de respuesta.**

**Fundamento:** No todos los parámetros requeridos para la ejecución de una acción de respuesta son proporcionados por el razonador. La información que contienen las alertas de intrusiones es insuficiente para ejecutar una respuesta. Consecuentemente, ciertos parámetros deben ser definidos por el administrador y obtenidos adecuadamente por el ejecutor de respuestas. Los parámetros fijos requeridos son:

- El nombre de la acción de respuesta, que como se mencionó en la sección 3.1.3, debe coincidir con el nombre de la propiedad *responseAction* de la ontología.
- Parámetros necesarios para contactar al o los componentes de seguridad.
- El grupo de identificadores de alertas de intrusiones (SIDS) ante los cuales se ejecuta una acción de respuesta.
- Parámetros propios de cada acción de respuesta:
    - El tiempo que tendrá efecto una acción de respuesta. Se puede deshabilitar un usuario durante 8 horas, o se puede añadir una regla de filtrado al firewall durante 2 horas por mencionar algunos ejemplos.
    - El criterio de bloqueo (modo). Útil para acciones de respuesta basada en red, y define si el bloqueo se realiza en función de la dirección IP

del atacante, de la víctima, de ambos o de la conexión completa (IPs, puertos y protocolo).

**CU07. Establecer comunicación.**

**Fundamento:** Como se ha mencionado en varios puntos de esta memoria, un IRS por naturaleza requiere interactuar con otros componentes de seguridad para ejecutar una acción de respuesta. Dichos componentes pueden ser: un firewall, para agregar reglas de filtrado; un router, para agregar listas de control de acceso; un hosts, para terminar un proceso, deshabilitar un usuario, respaldar un archivo hacia un servidor FTP o desplegar una red señuelo virtual. Por consiguiente, se debe establecer una comunicación ***segura y confiable*** con dicho componente de seguridad, a través de la cual se emite una petición de acción de respuesta.

**CU08. Cifrar y descifrar comunicación.**

**Fundamento:** Parece lógico pensar que este requerimiento está justificado por default. El presente trabajo trata acerca de los IDSs como un mecanismo para neutralizar un sinnúmero de ataques que atentan contra la confidencialidad, integridad y disponibilidad de un recurso; el módulo ejecutor de respuestas cae dentro de este grupo de recursos que también puede ser atacado, por consiguiente es necesario tomar las medidas pertinentes para protegerlo. En virtud de ello para que la emisión de una petición de acción de respuesta sea segura, la comunicación entre los involucrados debe ser cifrada. Además cabe recalcar en este punto, que el protocolo de comunicación debe ser *confiable.*

**CU09. Ejecutar acción de respuesta.**

**Fundamento:** Una vez que una petición de acción de respuesta ha sido construida, cifrada y emitida hacia el componente de seguridad respectivo; ésta debe ser ejecutada utilizando los ***medios y comandos propios de cada componente de seguridad***. Por ejemplo:

- Para agregar una regla de filtrado al firewall IPTables[5]: el *medio* a emplear puede ser un script para Bash[6], mientras que los comandos corresponden a una sintaxis propia de IPTables.
- Para agregar una lista de control de acceso a un router Cisco[7]: el medio a emplear puede ser una conexión telnet o ssh para ejecutar sobre la Interfaz

[5] http://www.netfilter.org/projects/iptables/
[6] http://www.gnu.org/software/bash/manual/bashref.html
[7] http://www.cisco.com/

de Línea de Comandos[8] (CLI) comandos propios de Cisco que puedan ser interpretados por el IOS.

- Para el despliegue de una honeynet usando VNX[9] (Virtual Network Over LinuX: el *medio* empleado puede ser un script Bash que despliega un escenario de red virtual escrito mediante un lenguaje de especificación VNX.
- Para deshabilitar un usuario sobre un host con S.O. Linux Ubuntu[10]: el *medio* empleado son comandos Bash con sintaxis propia de esta distribución de Linux.

Por consiguiente el sistema debe implementar un mecanismo tal que una respuesta pueda ejecutarse de *forma transparente* utilizando los *medios y comandos propios* de cada componente de seguridad. Además que sea *escalable,* es decir que se pueda desplegar fácilmente nuevas acciones de acciones de respuesta sobre diferentes componentes de seguridad.

**CU10. Verificar identidad del ejecutor.**

**Fundamento:** Su justificación guarda relación con lo definido en el caso de uso CU08. Antes de la ejecución de una acción de respuesta, se debe realizar la verificación del ejecutor que envía una petición de solicitud para evitar principalmente ataques de suplantación de identidad.

**CU11. Verificar lista blanca de ejecución.**

**Fundamento:** Una de las grandes amenazas de los sistemas de respuesta a intrusiones son los falsos positivos. Un falso positivo, en el contexto de un IDS, es definido como la generación inapropiada de un evento o alerta en respuesta a una detección de intrusión o ataque también equivocada [34]. En virtud de ello, los falsos positivos pueden ser generados intencionalmente con el objetivo de producir un ataque generalmente de denegación de servicio, como por ejemplo ejecutar una acción de bloqueo sobre el tráfico de usuarios legítimos. Por tal motivo, una petición de acción de respuesta no puede ejecutarse con total libertad y es necesario tomar alguna medida.

El AIRS basado en ontologías ya considera esta amenaza por medio del parámetro de *confianza del IDS* que es evaluado en la *métrica de reducción de daño (ver sección 2.4.3)*; no obstante, el ejecutor de respuestas debe definir una *lista blanca* que contenga un conjunto de componentes clasificados en base a *algún criterio*, sobre los cuales no se ejecutarán acciones de respuesta que resulten contraproducentes. Por ejemplo:

[8] http://www.cisco.com/warp/cpropub/45/tutorial.htm
[9] http://web.dit.upm.es/vnxwiki/index.php/Main_Page
[10] https://wiki.ubuntu.com/DocumentationTeam

- Lista blanca de hosts: en donde el *criterio* podría ser la dirección IP o el hostname. Un ejemplo de su utilidad es evitar que se bloquee el tráfico proveniente del Servidor de Nombres de Dominio (DNS), para ello se debería añadir la dirección IP o hostname del IDS.
- Lista blanca de SIDs: en donde el *criterio* es el identificador de intrusión. Snort[11] y OSSEC[12] un NIDS y HIDS respectivamente, generan un número que identifica a cada intrusión que podrían ser utilizados para generar una lista blanca. Por ejemplo: en el NIDS Snort una alerta, cuyo SID es 45608, es notificada constantemente disparando frecuentemente una acción de respuesta, pero se ha corroborado que no es dañina; aunque lo correcto sería ajustar el sensor de Snort, otra opción que puede resultar útil añadir el SID 45608 a la lista blanca para evitar este suceso.

Una vez establecidos los requerimientos generales, en la siguiente sección se realiza un bosquejo de la arquitectura del ejecutor de respuestas.

## 4.2 Esquema de arquitectura

En la Figura 14 se realiza una aproximación de la arquitectura del ejecutor de respuestas distribuido, que cubre los requisitos generales planteados en el caso de uso. Se identifican los siguientes componentes: los IDSs, el razonador que forma parte del AIRS basado en ontologías, el módulo central de ejecución, módulo de comunicación, los agentes de ejecución y los componentes de seguridad.

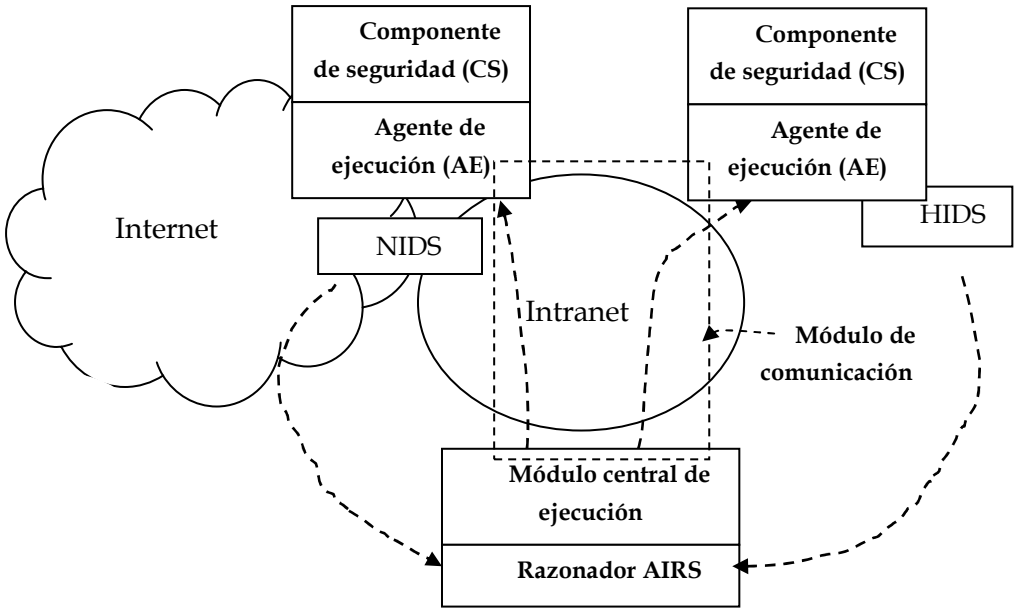



Figura 14. Esquema de arquitectura del ejecutor de respuestas.

[11] http://www.snort.org/
[12] http://www.ossec.net/

- **IDSs:** Son los sistemas de detección de intrusiones que monitorizan una fuente de datos y que notifican mediante el envío de alertas hacia el razonador del AIRS que una intrusión se ha detectado. En la sección 2.1.2 se hace una revisión de los diferentes tipos de IDSs existentes. Cualquiera de ellos se puede utilizar en la presente arquitectura. Provee la funcionalidad requerida por los casos de uso: CU01 y CU02.
- **Razonador AIRS:** En la Figura 5 está representada la arquitectura del AIRS basado en ontologías; en esta arquitectura se encuentra el razonador que infiere la respuesta óptima desde un catálogo de acciones de respuestas. Posteriormente invoca la acción de respuesta proporcionándole ciertos parámetros que son obtenidos de la alerta. En la sección 2.4 se aborda en detalle la arquitectura, proceso de inferencia y métricas de respuesta del AIRS basado en ontologías. Provee la funcionalidad requerida por los casos de uso: CU03 y CU04.
- **Módulo central de ejecución:** Luego de la invocación de una acción de respuesta, el módulo central de ejecución recibe los parámetros del *razonador AIRS* y también obtiene los parámetros fijos definidos por el administrador para cada acción de respuesta. Inmediatamente construye una solicitud de acción de respuesta que será enviado hacia el *agente de ejecución* adecuado a través del *módulo de comunicaciones*. Aborda los requerimientos definidos en los casos de uso: CU05 y CU06.
- **Módulo de comunicación:** Es el encargado del establecimiento de una comunicación entre el *módulo central de ejecució*n y *los agentes de ejecución*. Para ello debe hacer uso de un protocolo confiable. Además la solicitud de acción de respuesta es cifrada antes de su emisión; consecuentemente, en el otro extremo de la comunicación la solicitud de acción de respuesta es descifrada. Provee la funcionalidad requerida por los casos de uso: CU07 y CU08.
- **Agente de ejecución:** Recibe una solicitud de acción de respuesta desde el *módulo de comunicación* y verifica: primero, la autenticidad del emisor de la solicitud de acción de respuesta, y; segundo, que la solicitud no esté dirigida hacia uno de los componentes que se encuentra en *lista blanca*. Posteriormente utiliza el *medio y comandos* propios del componente de seguridad que incluye la acción de respuesta. Este módulo proporciona la funcionalidad demandada por los casos de uso: CU09, CU10 y CU11.
- **Componente de seguridad:** Representa el dispositivo que lleva a cabo la acción de respuesta real utilizando su interfaz de línea de comandos, tal que se altere su funcionamiento tan pronto como es ejecutado. Un router, firewall, servidor web, servidor FTP, gestor de usuarios, gestor de procesos son algunos ejemplos de ellos. Este módulo junto con el agente de ejecución proporciona la funcionalidad requerida por el caso de uso CU09.

## 4.3 Requisitos funcionales y no funcionales

En la Tabla 18 se muestra la asignación de los casos de uso a cada componente. Como se mencionó en la sección anterior los IDSs, el razonador AIRS y los compontes de seguridad son sistemas externos ya implementados; sin embargo imponen ciertos requisitos al ejecutor de respuestas que también se deben considerar. En virtud de ello, éste apartado tiene por objetivo establecer los requisitos funcionales asociados a cada componente, para lo cual se toma como referencia los casos de uso establecidos en la sección 4.1. Además, se establece los requisitos no funcionales.

| Componente | Casos de uso |
|---|---|
| **Sistemas de detección de intrusiones (IDSs)** | CU01, CU02 |
| **Razonador AIRS (RAIRS)** | CU03, CU04, CU05 |
| **Módulo central de ejecución (MCE)** | CU05, CU06 |
| **Módulo de comunicación (MC)** | CU07, CU08 |
| **Agente de ejecución (AE)** | CU09, CU10, CU11 |
| **Componente de seguridad (CS)** | CU09 |

**Tabla 18. Casos de uso por cada componente de la arquitectura propuesta.**

### 4.3.1 Requisitos funcionales

En la Tabla 19 se detallan los requisitos funcionales:

| Comp.* | C. de uso | Ref. | Requisito |
|---|---|---|---|
| IDSs | CU01 | F01 | Debe monitorizar la red usando IDSs basados en red e IDSs basados en hosts. |
| | CU02 | F02 | Debe emitir una alerta utilizando un formato de alerta en particular hacia el AIRS basado en ontologías. |
| RAIRS | CU03 | F03 | Debe receptar las alertas provenientes del IDS a través del *receptor de alertas* del AIRS basado en ontologías y parsear a la ontología correspondiente. |
| | CU04 | F04 | Debe inferir la respuesta óptima a la alerta de intrusión recibida. |
| | | F05 | Debe invocar la acción de respuesta usando el valor de la propiedad *responseAction* de la ontología del AIRS. |
| | | F06 | Debe proporcionarle al módulo central de ejecución, los siguientes parámetros: 1) Dirección IP origen (atacante), 2) Dirección IP destino (víctima), 3) Puerto destino, 4) Protocolo, 5) SID (signature ID), 6) protocolo; cuando se trata de una intrusión detectada por un NIDS. |
| | | F07 | Debe proporcionar al módulo central de ejecución, *al menos* los siguientes parámetros: 1) Dirección IP origen (atacante), 2) Dirección IP del host atacado, y 3) Nombre de usuario con el que se efectua la intrusión; cuando se trata de una intrusión detectada por un HIDS. |
| MCE | CU06 | F08 | Debe obtener la información de localización de los agentes de ejecución. |
| | | F09 | Debe tener la capacidad de formar grupos de agentes de ejecución sobre los que se ejecutará una acción de respuesta |
| | | F10 | Debe obtener los parámetros fijos asociados a cada respuesta, en donde el nombre que identifica a cada respuesta coincide con el parámetro *actionResponse* de la ontología del AIRS. |
| | | F11 | Debe tener la capacidad de agrupar varias acciones de respuesta y manejarla como si se tratara de una sola. |

| | | | |
|---|---|---|---|
| | | F12 | Debe construir el paquete de solicitud de acción de respuesta. |
| | | F13 | Debe emitir la solicitud al módulo de comunicaciones, proporcionándole la información necesaria para saber a dónde enviar. |
| MC | CU05 | F14 | Debe establecer una conexión con el o los agentes de ejecución definidos previamente por el MCE. |
| | CU07 | F15 | Debe cifrar la solicitud de acción de respuesta a emitirse hacia el agente de ejecución. |
| AE | CU09 | F16 | Debe obtener los parámetros necesarios de cada componente de seguridad con los que requiera interactuar. |
| | | F17 | Debe escuchar las solicitudes de respuesta emitidas por el módulo central de ejecución. |
| | | F18 | Debe recibir el paquete de solicitud de acción de respuesta. |
| | | F19 | Debe obtener el conjunto de comandos a ejecutar sobre el componente de seguridad. |
| | | F20 | Debe acceder a la interfaz de línea de comandos del componente de seguridad respectivo y ejecutarlos. |
| | | F21 | Debe tener la capacidad de ejecutar una acción de respuesta por un tiempo definido por el administrador, luego del cual la acción no tendrá efecto. |
| | | F22 | Si las *solicitudes de acción de respuesta son similares* y una respuesta ya se encuentra activa, entonces debe tener la capacidad de extender una acción de respuesta. |
| | CU10 | F23 | Debe verificar la identidad del Módulo central de ejecución. |
| | CU11 | F24 | Debe obtener la lista blanca de entidades sobre las que no se ejecuta una acción de respuesta. |
| | | F25 | Debe verificar que la solicitud de acción de respuesta no incluya a una entidad incluida en la lista blanca. |
| CS | CU09 | F26 | Debe proporcionar un medio de acceso a la interfaz de línea de comandos. |

**Tabla 19. Requisitos funcionales. *IDSs: Sistemas de detección de intrusiones, RAIRS: Razonador AIRS, MCER: Módulo central de ejecución de respuestas, MC: Módulo de comunicación, AE: Agente de Ejecución, CS: Componente de seguridad.**

### 4.3.2 Requisitos no funcionales

En la Tabla 20 se describen los requisitos no funcionales a considerar en el diseño e implementación del ejecutor de respuestas:

| **Tipo** | **Comp.*** | **Ref.** | **Requisito** |
|---|---|---|---|
| Externos de interoperabilidad | MCER | NF01 | Debe ser implementado en Java, o al menos un submódulo de integración, dado que el ReasonerAIRS está desarrollado en este lenguaje. |
| | | NF02 | Debe tomar como referencia la información de las alertas de intrusiones generadas por el NIDS Snort y el HIDS OSSEC. |
| | | NF03 | Debe usar el valor de la propiedad *actionResponse* de la ontología del AIRS como nombre de la acción de respuesta. |
| Portabilidad | ME | NF03 | Debe operar sobre el S.O. Linux y dejar abierta la posibilidad de extenderlos a otros S.O. |
| Escalabilidad | ME | NF04 | El conjunto de comandos a ejecutar sobre el componente de seguridad deben ser encapsulados, como lógica de control |

| | | | |
|---|---|---|---|
| | | | independiente de las funciones principales del Agente de ejecutoción, de tal forma que se garantice un sencillo despliegue de nuevas acciones de respuesta sobre diferentes componentes de seguridad. |
| Seguridad | MC | NF05 | Debe garantizar la confidencialidad por medio del cifrado de la solicitud de acción de respuesta. |
| | ME | NF06 | Debe comprobar la identidad del ejecutor de una acción para evitar ataques por suplantación de identidad. |
| Confiabilidad | MC | NF06 | Debe garantizar la entrega de la solicitud de acción de respuesta. |
| Mantenimiento | MCER | NF07 | Debe tener la capacidad de almacenar los logs de operación. |
| | AE | NF08 | Debe tener la capacidad de almacenar los logs de operación. |

**Tabla 20. Requisitos no funcionales. * MCER: Módulo central de ejecución de respuestas, MC: Módulo de comunicación, AE: Agente de Ejecución.**

# 5 Diseño de la arquitectura

Una vez que se han definido los requisitos funcionales y no funcionales del ejecutor de respuestas, descritos en las Tablas 19 y 20 respectivamente, este capítulo tiene por objetivo reseñar los aspectos más relevantes del diseño arquitectónico de dicho ejecutor junto con el diseño procedimental de cada uno de sus módulos.

No obstante, previo al diseño e implementación del ejecutor de respuestas se ha realizado una revisión de herramientas Open Source, cuya arquitectura podría cumplir con los requerimientos antes planteados. El objetivo de esta revisión fue reconocer ciertas pautas de diseño de la arquitectura y en la medida de lo posible identificar aquellos componentes que puedan ser reutilizados en la implementación del prototipo.

Se evaluaron tres herramientas: SnortSam[13], FwSnort[14] y Snort_Inline[15]. Las tres herramientas tienen capacidad de *respuesta activa basada en red* y responden ante alertas emitidas por un NIDS Snort[16], también Open Source. El criterio principal de evaluación fue la capacidad de la aplicación para ejecutar una acción de respuesta de forma distribuida, es decir que tenga la capacidad de interactuar con varios componentes de seguridad. A continuación se realiza una descripción breve de cada una de estas herramientas.

- SnortSam es un *sistema de respuesta activa* que interactúa con firewalls comerciales y Open Source bloqueando direcciones IP. Esta adaptado como un plugin de salida de Snort para ejecutar una acción de bloqueo *exclusivamente* ante alertas generadas por dicho IDS. Para ello, despliega componentes remotos que aceptan solicitudes de bloqueo enviados a través de una sesión TCP cifrada.
- FWSnort aprovecha la capacidad del firewall NetFilter[17], que se encuentra en los sistemas Linux, para filtrar paquetes basados en los campos de las cabeceras de red, transporte y aplicación de un paquete que circula a través de él. Para ello, traduce las reglas definidas en los ficheros de configuración de Snort a su regla equivalente en el firewall NetFilter. FWSnort obligatoriamente debe ser desplegado dentro de NetFilter ya que es necesario estar *en línea* con el tráfico de las redes que protege el Firewall; consecuentemente, por definición se podría decir que FWSnort funciona como un IPS más que como un IRS.

[13] http://www.snortsam.net/
[14] http://www.cipherdyne.org/fwsnort/
[15] http://snort-inline.sourceforge.net/oldhome.html
[16] http://www.snort.org/
[17] http://www.netfilter.org/

- Snort_Inline es una versión modificada de Snort que al igual que FWSnort interactúa con el firewall NetFilter; sin embargo, no traduce las reglas definidas por Snort, en su lugar construye una nueva regla de filtrado ante una intrusión detectada por el NIDS Snort. Tiene la capacidad de alterar o descartar los paquetes en tiempo real, conforme fluyen a través de él; por tal razón se requiere que snort_inline se encuentre *en línea* con el tráfico a proteger. Por definición cae también en la categoría de un IPS en lugar de un IRS.

| Herramienta | Distribuida | Respuestas basadas en red | Respuestas basadas en host |
|---|---|---|---|
| **SnortSam** | Si | Bloqueo Firewall | No |
| **FwSnort** | No | Bloqueo Firewall | No |
| **Snort_Inline** | No | Bloqueo Firewall | No |

**Tabla 21. Herramientas Open Source de respuesta activa .**

Las tres herramientas pueden ejecutar una *respuesta activa de protección*, que consiste en el bloqueo de la conexión del atacante por medio de un firewall. Sin embargo existe una diferencia entre ellos:

- Tanto *FWSnort* como *Snort_Inline* son IPSs, por tal motivo deben estar ubicados en línea con el flujo de transacciones que incluyen a la potencial intrusión. En este caso, al tratarse de dos herramientas que operan con un NIDS, debe funcionar sobre un *dispositivo inline*, es decir deben ser un salto en la ruta que atraviesa un paquete de red a medida que ingresa o sale de la red o debe actuar como un puente entre dos segmentos de red Ethernet. Su ubicación es crucial para neutralizar un ataque; sin embargo, no interactúan con otros componentes de seguridad dada su naturaleza de IPS. Consecuentemente, estás herramientas no despliegan *otras respuesta activas* interactuando de forma distribuida con otros componentes de seguridad, razón por la cual no aportarían mayor funcionalidad al prototipo a implementar y no han sido consideradas.
- Por otro lado, un IRS cumple con las mismas funciones que un IPS; sin embargo, no se limita a ejecutar acciones de bloqueo sobre un *dispositivo inline*. Un IRS dispone de un *conjunto de respuestas* activas y pasivas que se pueden ejecutar sobre otros componentes de seguridad y la selección de una de ellas se hace en función de un análisis previo que evalúa el costo de ejecutar dicha respuesta. Aunque *SnortSam* no puede ser considerado un IRS dado que solo cuenta con *una respuesta activa de protección* y no evalúa el costo de ejecutar una respuesta (ejecuta la respuesta tan pronto que Snort detecta una intrusión); esta herramienta puede ser de mucha utilidad para la

implementación del ejecutor de respuestas del AIRS basado en ontologías dada su arquitectura distribuida y basada en plugins, y se aborda con mayor detalle en la siguiente sección.

Este capítulo está dividido en 2 partes:

1. Se describe los aspectos más relevantes de la herramienta SnortSam, que servirán para identificar aquellos componentes que pueden ser reutilizados en el diseño e implementación del ejecutor de respuestas.
2. Se realiza el diseño de la arquitectura del ejecutor de respuestas considerando 3 aspectos: la organización del sistema, su descomposición modular y el esquema de control genérico a utilizar. Se presenta también los aspectos más relevantes del diseño procedimental de cada uno de los módulos que conforman la arquitectura.

## 5.1 SnortSam

SnortSam es una herramienta Open Source bajo licencia GPL, que funciona bajo una arquitectura cliente-servidor. Opera como un plugin para Snort permitiendo el bloqueo automático de direcciones IP sobre firewalls open source y comerciales. SnortSam está constituido de dos componentes:

- Un plugin de salida para Snort que es implementado como un parche al código fuente de Snort.
- Un agente que corre sobre un firewall basado en host y está a la espera de comandos enviados desde el plugin de salida que se transmiten a través de la red.

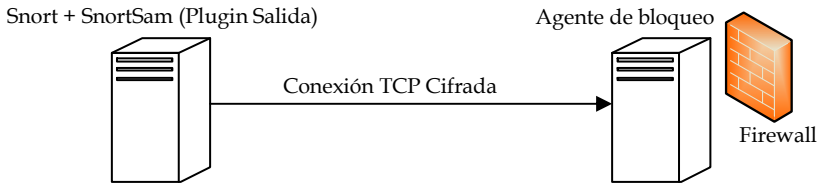



**Figura 15. Componentes de SnortSam.**

El agente de bloqueo es el responsable de interactuar con el firewall para bloquear dinámicamente las direcciones IP que arriban desde el sensor Snort que ha detectado un ataque. Los firewalls con los que puede interactuar SnortSam actualmente son: FW-1 de CheckPoint, PIX de Cisco, Netscreen de Juniper, IPFilter, IPTables, IPChains, WatchGuard, 8signs, MS ISA Server, EBTables y MS ISA Server [55].

Uno de los grandes inconvenientes encontrados al tratar con SnortSam fue la falta de documentación. En virtud de ello, se procedió a analizar su funcionamiento y código fuente con dos propósitos:

1. Determinar una aproximación de la arquitectura propuesta;
2. Determinar qué requerimientos funcionales y no funcionales del ejecutor de respuestas, establecidos en el capítulo 4, son proporcionados por SnortSam.

Luego de llevar a cabo ambas tareas se identificarán los módulos que pueden ser reutilizados en el diseño e implementación del ejecutor de respuestas del AIRS basado en ontologías.

### 5.1.1 Arquitectura

En la Figura 16 se observa una aproximación de la arquitectura de SnortSam. Se basa en una arquitectura distribuida cliente servidor conformada por dos componentes principales: un plugin de salida para Snort y un agente de bloqueo.

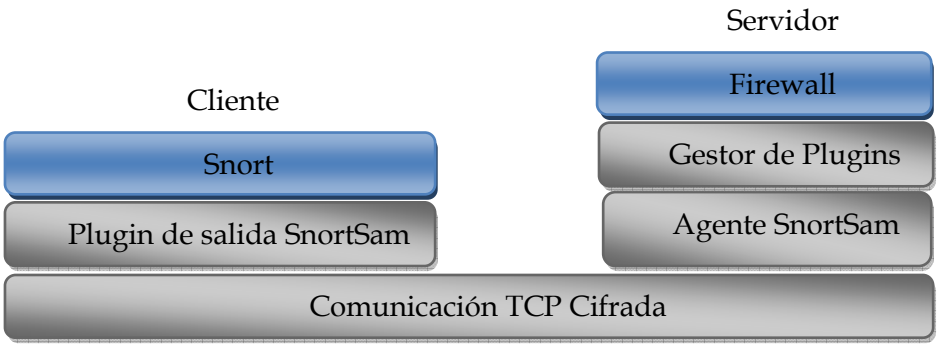



Figura 16. Arquitectura de SnortSam.

#### *Plugin de salida para Snort*

El cliente lo conforma el sensor Snort que tiene registrado el plugin de salida SnortSam. Cuando Snort detecta una intrusión, el plugin SnortSam envía un mensaje de bloqueo con la dirección IP del atacante a través de una comunicación TCP cifrada.

Para poner en marcha el lado del cliente se requiere la modificación de dos archivos de configuración de Snort: el archivo de configuración principal y las reglas individuales definidas por Snort. Cuando una intrusión coincide con alguna de las reglas que han sido configuradas para responder con SnortSam, el plugin de salida se comunica con el agente que está corriendo sobre un firewall, a través del puerto TCP 898. La comunicación es cifrada por medio de una contraseña que ha sido definida en ambos extremos del canal de comunicación.

- En primera instancia se tiene que indicarle a Snort cómo localizar al agente; para ello se debe definir una sentencia de activación, indicando la dirección del agente, puerto y contraseña de cifrado, en el archivo de configuración principal.
    - *output alert_fwsam: 192.168.10.l:1200/pa5W02r9*

- Posteriormente, se especifica a Snort ante qué reglas se dispara una acción de bloqueo; para ello se debe agregar a cada regla la opción *fsam* junto con otros parámetros adicionales:
    - *alert tcp any any -> $10.1.1.100 8080 (msg:"WEB-CGI /wwwboard/passwd.txt access"; flow:to_server,established; uricontent:"/wwwboard/passwd.txt"; nocase; reference:arachnids,463; reference:eve,CVE 1999-0953; referencemessus,10321; referencerbugtraq,649; classtype:attempted-recon; sid:807; rev:7;* ***fwsam: src, 2 hour;)***

  En este ejemplo se ha añadido la opción fwsam a la regla con SID 807 que se encuentra en el fichero de reglas *web-cgi.rules*. De esta manera se bloquean todas las direcciones IP, por un período de dos horas, ante la alerta de una intrusión que coincida con la regla anteriormente definida.

### *Agente de bloqueo*

El servidor, que despliega una arquitectura basada en plugins, está constituido por el *agente de bloqueo SnortSam y el gestor de plugins*; ambos trabajan de forma conjunta para ejecutar bloqueos sobre firewalls de distintos fabricantes, sean estos comerciales u open source, de la siguiente manera:

- Si Snort detecta un ataque que coincide con cualquier regla que tiene definida la opción *fwsam*, se establece una sesión TCP cifrada con el agente de bloqueo y se envía un mensaje que contiene la dirección IP del atacante que originó la alerta junto con un valor que define el tiempo que tendrá efecto el bloqueo.
- El agente está escuchando de forma continua a través del puerto TCP 898 (por defecto), esperando los mensajes enviados por el plugin de salida.
    - El agente SnortSam cuenta con uno o varios plugins que han sido registrados a través del gestor de plugins (Ejm: Plugin IPTables, Plugin PIX, Plugin Screenet, etc.)
- Cuando arriba un mensaje de bloqueo, el agente SnortSam ejecuta el bloqueo de una IP sobre *todos* los plugins que tenía registrados.
- El agente de bloqueo mantiene el estado de todas las direcciones IP bloqueadas, dentro del fichero */var/log/snortsam.state*. Este fichero sirve para evitar reglas de bloqueo duplicadas si el agente ha sido detenido o reiniciado por alguna razón.
- El gestor de plugins maneja un fichero de *registro de plugins*, en el que se referencian las rutinas, específicas de cada firewall, que se ejecutarán ante la recepción de un mensaje de bloqueo. Este hecho le proporciona gran flexibilidad, ya que es posible implementar nuevos plugins que interactúen con otros firewalls, sin modificar otros módulos que componen SnortSam.

### 5.1.2 Funcionalidad

Esta sección tiene por objetivo identificar los requerimientos funcionales y no funcionales, que fueron establecidos en el capítulo 4, a los que da cumplimiento SnortSam. Se tiene el propósito de poder reutilizar ciertos módulos tanto en el diseño como en la implementación del ejecutor de respuestas.

| Comp.* | Ref. | Requisito | SnortSam | Observación |
|---|---|---|---|---|
| IDSs | F01 | Debe monitorizar la red usando IDSs basados en red e IDSs basados en hosts. | ½ | Sólo NIDS. |
| | F02 | Debe emitir una alerta utilizando un formato de alerta en particular hacia el AIRS basado en ontologías. | No | |
| RAIRS | F03 | Debe receptar las alertas provenientes del IDS a través del *receptor de alertas* del AIRS basado en ontologías. | No | |
| | F04 | Debe inferir la respuesta óptima a la alerta de intrusión recibida. | No | |
| | F05 | Debe invocar la acción de respuesta usando el valor de la propiedad *responseAction* de la ontología del AIRS. | No | |
| | F06 | Debe proporcionarle al módulo central de ejecución, los siguientes parámetros: 1) Dirección IP origen (atacante), 2) Dirección IP destino (víctima), 3) Puerto destino, 4) Protocolo, 5) SID (signature ID), 6) protocolo; cuando se trata de una intrusión detectada por un NIDS. | No | |
| | F07 | Debe proporcionar al módulo central de ejecución, *al menos* los siguientes parámetros: 1) Dirección IP origen (atacante), 2) Dirección IP del host atacado, y 3) Nombre de usuario con el que se efectúa la intrusión; cuando se trata de una intrusión detectada por un HIDS. | No | |
| MCE | F08 | Debe obtener la información de localización de los agentes de ejecución. | 1/2 | Está ajustado a los archivos de configuración de Snort. |
| | F09 | Debe tener la capacidad de agrupar grupos de agentes de ejecución sobre los que se ejecutará una acción de respuesta | No | |
| | F10 | Debe obtener los parámetros fijos asociados a cada respuesta, en donde el nombre que identifica a cada respuesta coincide con el parámetro *actionResponse* de la ontología del AIRS. | No | |
| | F11 | Debe tener la capacidad de agrupar varias acciones de respuesta y manejarla como si se tratara de una sola. | No | |
| | F12 | Debe construir el paquete de solicitud de acción de respuesta. | 1/2 | Se deben agregar nuevos campos. |
| | F13 | Debe emitir la solicitud al módulo de comunicaciones, proporcionándole la información necesaria para saber a dónde enviar. | No | |
| MC | F14 | Debe establecer una conexión con el o los agentes de ejecución definidos previamente por el MCE. | Si | |
| | F15 | Debe cifrar la solicitud de acción de respuesta a emitirse hacia el agente de ejecución. | Si | |

| | | | | |
|---|---|---|---|---|
| AE | F16 | Debe obtener los parámetros necesarios de cada componente de seguridad con los que requiera interactuar. | 1/2 | Únicamente opera sobre firewalls. |
| | F17 | Debe escuchar las solicitudes de respuesta emitidas por el módulo central de ejecución. | Si | |
| | F18 | Debe recibir el paquete de solicitud de acción de respuesta. | Si | |
| | F19 | Debe obtener el conjunto de comandos a ejecutar sobre el componente de seguridad. | 1/2 | Únicamente trabaja con firewalls. |
| | F20 | Debe acceder a la interfaz de línea de comandos del componente de seguridad respectivo y ejecutarlos. | No | |
| | F21 | Debe tener la capacidad de ejecutar una acción de respuesta por un tiempo definido por el administrador, luego del cual la acción no tendrá efecto. | Si | |
| | F22 | Si las *solicitudes de acción de respuesta son similares* y una respuesta ya se encuentra activa, entonces debe tener la capacidad de extender una acción de respuesta. | Si | |
| | F23 | Debe verificar la identidad del Módulo central de ejecución. | Si | |
| | F24 | Debe obtener la lista blanca de entidades sobre las que no se ejecuta una acción de respuesta. | Si | |
| | F25 | Debe verificar que la solicitud de acción de respuesta no incluya a una entidad incluida en la lista blanca. | Si | |
| CS | F26 | Debe proporcionar un medio de acceso a la interfaz de línea de comandos. | Si | |

**Tabla 22. Requisitos funcionales que cumple SnortSam.**

| **Tipo** | **Comp.*** | **Ref.** | **Requisito** | **SnortSam** |
|---|---|---|---|---|
| Externos de interoperabilidad | MCER | NF01 | Debe ser implementado en Java, o al menos un submódulo de integración, dado que el ReasonerAIRS está desarrollado en este lenguaje. | No |
| | | NF02 | Debe tomar como referencia la información de las alertas de intrusiones generadas por el NIDS Snort y el HIDS OSSEC. | No |
| | | NF03 | Debe usar el valor de la propiedad *actionResponse* de la ontología del AIRS como nombre de la acción de respuesta. | No |
| Portabilidad | ME | NF03 | Debe operar sobre el S.O. Linux y dejar abierta la posibilidad de extenderlos a otros S.O. | Si |
| Escalabilidad | ME | NF04 | El conjunto de comandos a ejecutar sobre el componente de seguridad deben ser encapsulados, como lógica de control independiente de las funciones principales del Agente Ejecutor, de tal forma que se garantice un sencillo despliegue de nuevas acciones de respuesta sobre diferentes componentes de seguridad. | Si |

| | | | | |
|---|---|---|---|---|
| Seguridad | MC | NF05 | Debe garantizar la confidencialidad por medio del cifrado de la solicitud de acción de respuesta. | Si |
| | ME | NF06 | Debe comprobar la identidad del ejecutor de una acción para evitar ataques por suplantación de identidad. | Si |
| Confiabilidad | MC | NF06 | Debe garantizar la entrega de la solicitud de acción de respuesta. | Si |
| Mantenimiento | MCER | NF07 | Debe tener la capacidad de almacenar los logs de operación. | No |
| | AE | NF08 | Debe tener la capacidad de almacenar los logs de operación. | Si |

**Tabla 23. no funcionales a los que da cumplimiento SnortSam.**

Una vez establecida una aproximación de la arquitectura y funcionalidad provista por SnortSam se pueden obtener las siguientes conclusiones:

- SnortSam puede ser considerado una aplicación de respuesta activa, más no un IRS, que lleva a cabo una respuesta de protección mediante el bloqueo de la dirección IP del atacante sobre un firewall.
- Está acoplado como un plugin de salida de Snort; por consiguiente, no puede ser utilizado directamente por el AIRS basado en ontologías.
- Provee una arquitectura distribuida, requerimiento necesario del ejecutor de respuestas para interactuar con varios componentes de seguridad.
- Los agentes de bloqueo manejan una arquitectura basada en plugins, proporcionando flexibilidad a la hora de desplegar una nueva acción de bloqueo.

En virtud de lo antes mencionado, se puede considerar reutilizar el módulo de comunicación, el agente y el gestor de plugins de SnortSam, para diseñar e implementar el ejecutor de respuestas del AIRS basado en ontologías.

## 5.2 Propuesta de arquitectura

En esta sección se presenta la arquitectura del ejecutor de respuestas para el AIRS basado en ontologías. El objetivo es mostrar una visión general del ejecutor, estableciendo un marco de control y comunicación entre los módulos que conforman dicha arquitectura. Puesto que el diseño arquitectónico es un proceso en el que se intenta establecer una organización del ejecutor de respuestas que satisfaga los requerimientos funcionales y no funcionales, se ha divido el diseño de la arquitectura considerando tres aspectos: organización del sistema, descomposición modular y esquema de control.

### 5.2.1 Organización del sistema

Se propone una organización del sistema basada en el modelo arquitectónico cliente-servidor, cuyo diagrama de despliegue se muestra en la Figura 17. La ventaja

más importante que conlleva a utilizar esta arquitectura es su naturaleza distribuida, que es un requisito inherente del ejecutor de respuestas, dada la necesidad de contar con múltiples agentes distribuidos a los que se les pueda solicitar la ejecución de una acción de respuesta.

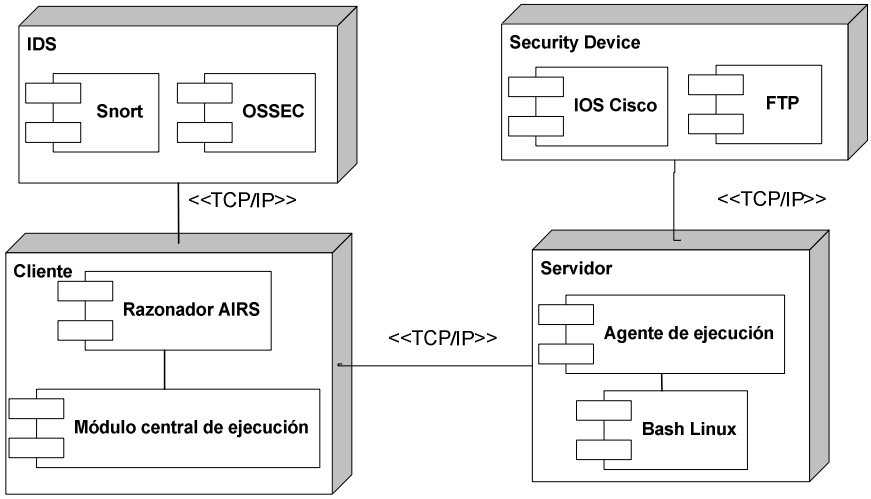



**Figura 17. Arquitectura cliente-servidor del ejecutor de respuestas.**

- Los *IDSs*, actúan como sistemas externos que se encuentran distribuidos en la red, por ejemplo Snort y OSSEC son IDSs open source que envían alertas de intrusiones hacia el Razonador AIRS, para ello emplean protocolos TCP/IP.
- El Razonador AIRS actúa también como un sistema externo y reside en el mismo lugar que el módulo central de ejecución.
- El *módulo central de ejecución* en este caso funciona como un cliente ligero que solicita un único servicio a los *agentes de ejecución*, que es la ejecución, propiamente dicha, de una acción de respuesta. El *módulo central de ejecución* de antemano conoce las direcciones IP de los *agentes de ejecución* a quienes solicita la ejecución de una acción de respuesta mediante el envío de una solicitud, utilizando el protocolo de transporte TCP.
- El *agente de ejecución*, quien actúa como servidor, está escuchando constantemente por un puerto TCP las peticiones que arriban desde el *módulo central de ejecución*. Además el *agente de ejecución* dependiendo del tipo de *componente de seguridad*, puede interactuar con éste de dos formas:
    1. **Ejecución de forma local**: Cuando el *componente de seguridad* y el *agente de ejecución* se despliega en el mismo dispositivo. Por ejemplo en la Figura 17, si el *componente de seguridad IPTables* está sobre un host con S.O. Ubuntu, entonces el *agente de ejecución* ejecuta una acción de respuesta directamente por medio de la interfaz de línea de comandos de Bash Linux.

2. **Ejecución de forma remota:** Cuando un *agente de ejecución* no puede ser instalado sobre el dispositivo que contiene el *componente de seguridad*, al tratarse de un dispositivo dedicado. Por ejemplo en la Figura 17, para ejecutar una acción de respuesta sobre un router Cisco, el *agente de ejecución* debe establecer previamente una conexión telnet o ssh con el dispositivo remoto para acceder a la interfaz de línea de comandos.

### 5.2.2 Descomposición modular

El estilo de descomposición modular está orientado a una agrupación de funciones afines, en donde cada módulo procesa los datos de entrada y proporciona los datos de salida respectivos. Cada módulo funciona de manera independiente, y para su comunicación basta con conocer las interfaces de entrada y salida respectivas.

En base a los requerimientos funcionales y no funcionales, detallados en las Tablas 19 y 20 respectivamente, se han definido 6 módulos: *Razonador AIRS e IDSs, módulo central de ejecución, módulo de comunicación, agente de ejecución, gestor de plugins y los componentes de seguridad.* La arquitectura desde la perspectiva de la descomposición modular se presenta en la Figura 18.

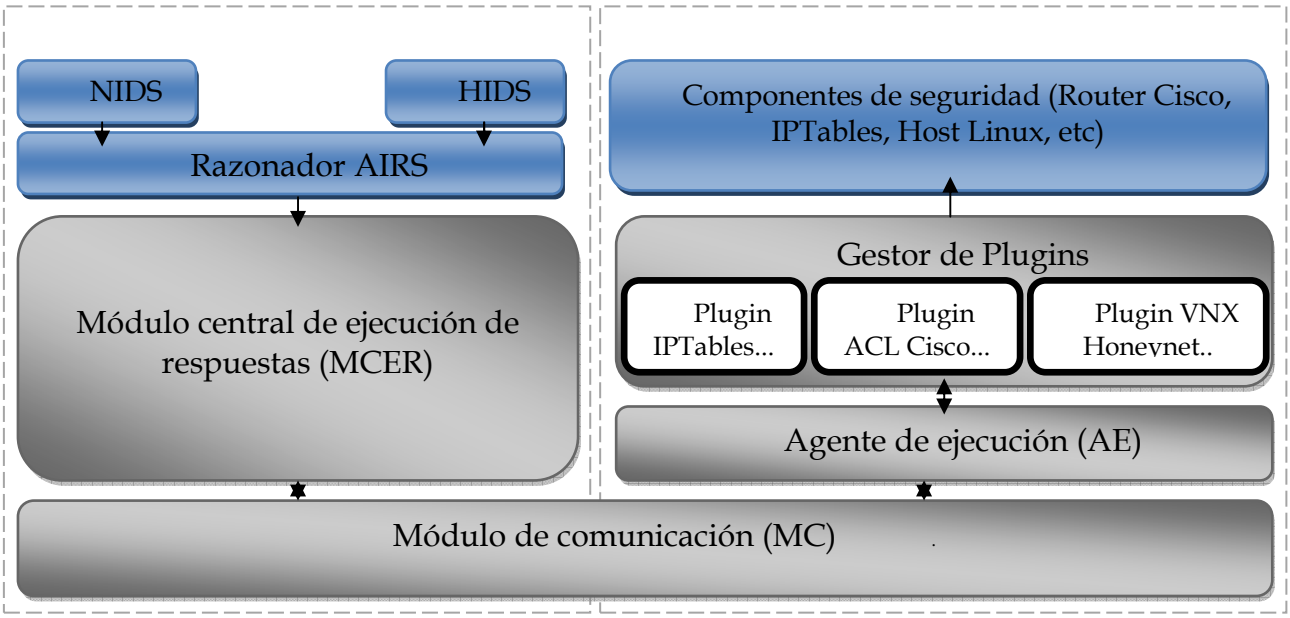



**Figura 18. Arquitectura del ejecutor de respuestas según su descomposición modular.**

Cómo se puede apreciar en la Figura 18, la arquitectura del ejecutor de respuestas según su descomposición modular no difiere en gran medida con la organización del sistema establecida en la sección anterior. La única diferencia radica en la inclusión del módulo *gestor de plugins,* adoptado de la arquitectura de SnortSam. El gestor de plugins

cumple un rol importante dentro de la arquitectura propuesta, ya que permite dar cumplimiento al requisito no funcional de escalabilidad (NF04) del agente de ejecución.

Es importante recalcar que el *módulo de comunicaciones, el agente de ejecución y el gestor de plugins,* propios de SnortSam, serán reutilizados en el prototipo del ejecutor de respuestas. Por supuesto, varias modificaciones han sido realizadas sobre dichos componentes para adaptarlas a la arquitectura que se muestra en la Figura 18. Dichas modificaciones son abordadas en el capítulo 6.

Ahora bien, para que el ejecutor de respuestas pueda trabajar como un sistema completo cada uno de los módulos deben ser controlados de tal forma que sus funciones se lleven a cabo en el momento adecuado, de acuerdo a una secuencia específica. Por consiguiente, en la Figura 19 se presenta un diagrama de actividades que define un estilo de control genérico empleado por el ejecutor de respuestas.

#### *IDSs*

Pueden ser IDSs basados en hosts o IDSs basados en red, quienes tras detectar una intrusión notifican mediante una alerta al AIRS basado en ontologías. Una revisión completa de los IDSs se presenta en la sección 2.1 de la presente memoria.

Los IDSs, dentro de la arquitectura propuesta, actúan como sistemas externos y son quienes inician el proceso, más no son los que deciden ejecutar una respuesta, como sucede con SnortSam descrito en la sección 5.1. La razón es que las alertas emitidas son analizadas por el *AIRS basado en ontologías* y luego de un proceso de inferencia en base a ciertas métricas de respuesta se elige la respuesta óptima; el *AIRS basado en ontologías* fue descrito detalladamente en la sección 2.4.

#### *Razonador AIRS*

Una alerta de intrusión es recibida por medio del *receptor de alertas del AIRS basado en ontologías*, quien se encarga del parsing de las alertas a una instancia de *alerta* de la ontología del AIRS. Posteriormente se realiza el proceso de inferencia en base a las métricas de respuesta, seleccionando la respuesta óptima a ejecutarse.

Una vez que el *razonador AIRS* ha inferido la respuesta óptima, invoca dicha respuesta usando el valor de la propiedad *actionResponse* de la ontología del AIRS; en dicha invocación se envían los parámetros requeridos por una acción de respuesta específica que son obtenidos de la alerta de intrusión o utilizando alguna herramienta automatizada.

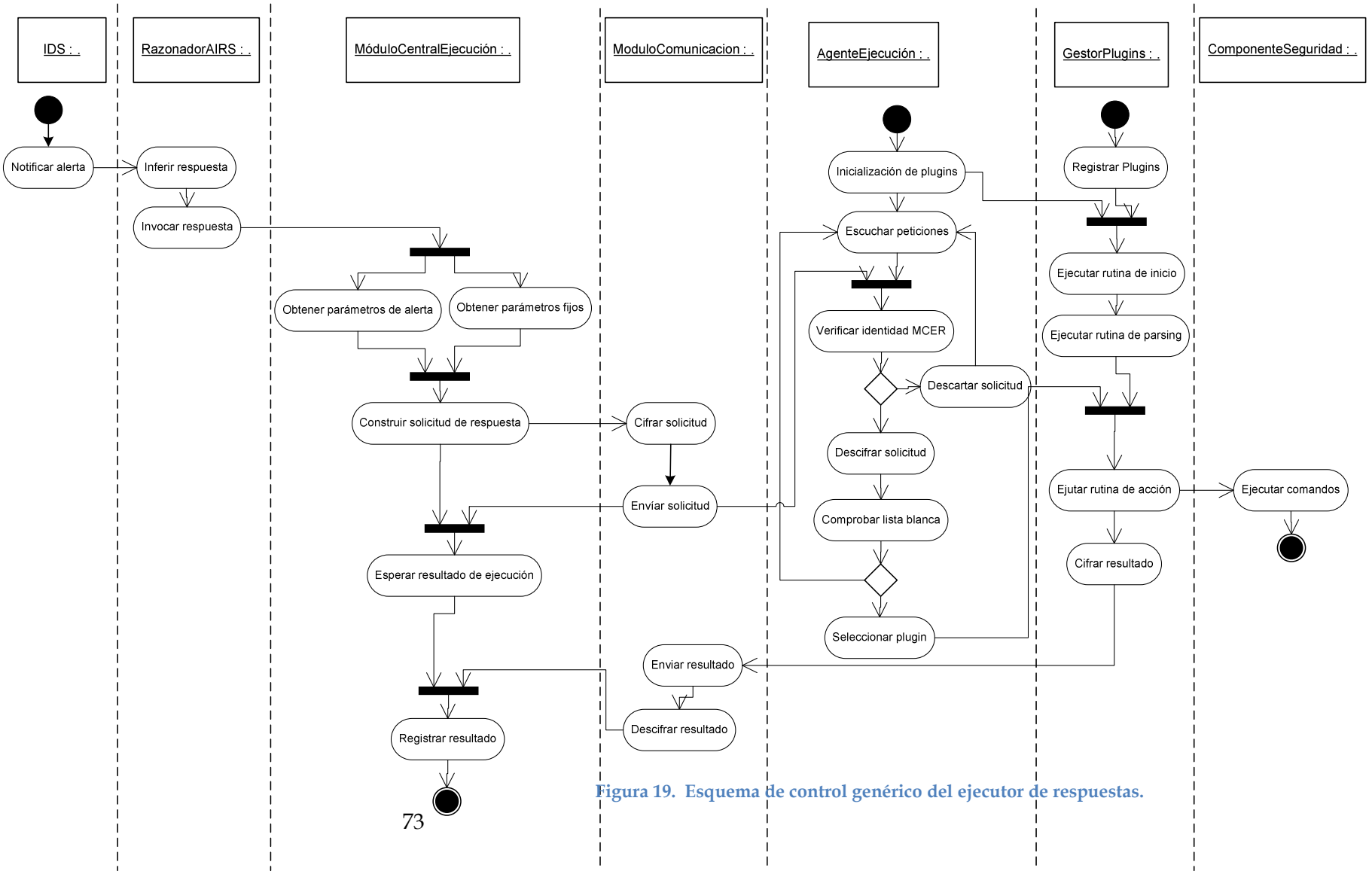



Figura 19. Esquema de control genérico del ejecutor de respuestas.

Es importante señalar que el r*azonador AIRS* es quien debe garantizar y proporcionar los **parámetros necesarios relacionados al ataque** que permitan ejecutar una acción de respuesta. Por ejemplo:

- Una acción de respuesta con nombre *DenegarConexion,* que bloquea el tráfico asociado a una conexión específica sobre un firewall, debe invocar la acción de respuesta proporcionando las *direcciones IP origen y destino, puertos y protocolo* empleado en la conexión.
- Por su parte, una acción de respuesta con nombre *DeshabilitarUsuario,* que deshabilita a un usuario que intenta perpetrar un ataque, debe invocar la acción de respuesta proporcionando *la dirección IP del host atacado y el nombre de usuario.*

Los parámetros pueden ser obtenidos desde las alertas o utilizando otras herramientas como nagios[18], net-snmp[19], etc. En cualquier caso, el *módulo central de ejecución*, que se describe en la siguiente sección, asume que los parámetros relacionados a la intrusión serán suministrados por el razonador AIRS.

### *Módulo Central de Ejecución de Respuesta (MCER)*

El *módulo central de ejecución de respuesta* se encarga básicamente de dos tareas: construir una solicitud de respuesta e identificar el o los agentes de ejecución a donde se enviará la solicitud de respuesta.

En la Figura 20 se observa el modelo de datos del MCER. Para construir una solicitud de acción de respuesta, el MCER recibe los parámetros de dos fuentes:

- Desde el razonador AIRS, quien envía los parámetros obtenidos desde las alertas u obtenidos mediante herramientas adicionales. *Ver Tabla ResponseActionsParams.*
- De información fija proporcionada por el administrador, que incluye:
  - *Grupo de agentes de ejecución*, hacia donde se envía una solicitud de respuesta. Cada agente de ejecución contiene información para su localización: dirección IP, puerto y contraseña de cifrado. *Ver Tabla ExecutoAgent y GroupExecutorAgents.*
  - *Grupo de Identificadores de alerta (Signatures ID),* que determina ante qué intrusiones, identificadas por su SID, es posible ejecutar una acción de respuesta. *Ver Tabla SidsGroup.*

[18] http://www.nagios.org/
[19] http://www.net-snmp.org/

- *Identificador de plugin,* asegura que sobre el otro extremo de la comunicación, es decir sobre el agente de ejecución, se ejecute únicamente el plugin que coincida con este valor. *Ver Tabla Plugin.*
- *Duración de la respuesta (duration),* determina el intervalo de tiempo que tendrá efecto una acción de respuesta.
- *Modo de ejecución (mode),* determina el criterio en base al cual se ejecuta una acción de respuesta basada en red. En función del tráfico de entrada (in), salida (out), entrada y salida (inout) o de esa conexión específica (conn). Esta opción se hereda de SnortSam.
- *Identificación del atacante (who),* indica qué dirección IP de la alerta es considerado el atacante: origen o destino. Evita ejecutar equivocadamente una acción de respuesta y ocasionar una denegación de servicio en nuestra red.
- *Identificación de respuesta compuesta (composed),* indica si la respuesta está constituida por varias acciones.
- *Respuestas (responses),* solo es utilizada si el campo anterior es verdadero y hace referencia a las respuestas simples conforman la respuesta compuesta.

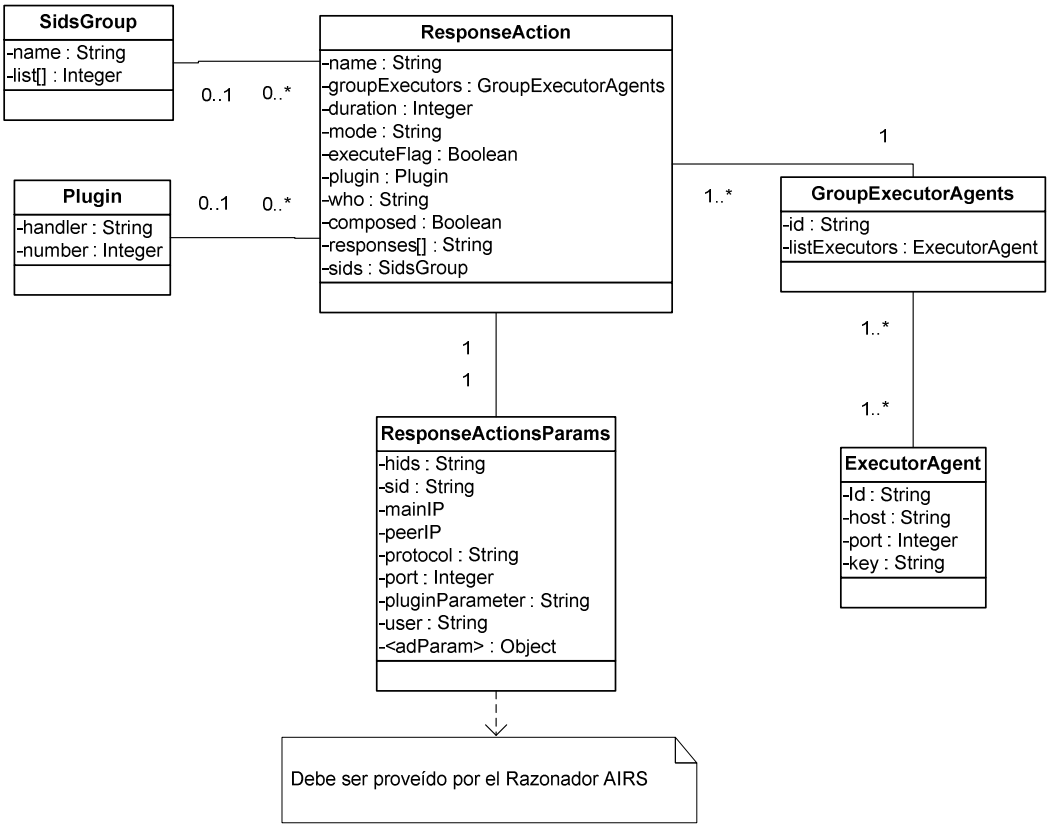



**Figura 20. Modelo de datos del módulo central de ejecución de respuesta.**

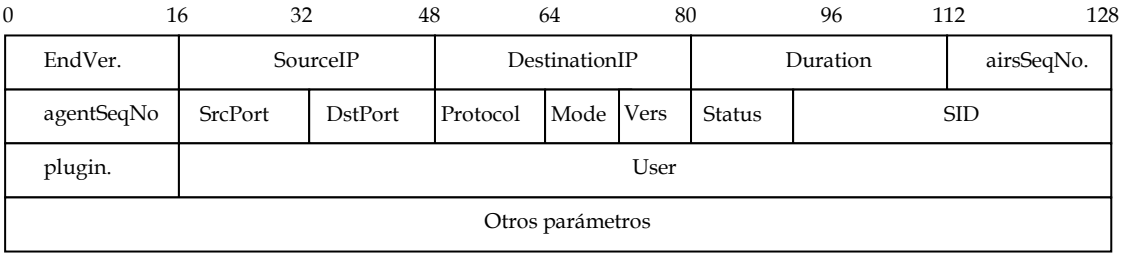
### *Módulo de Comunicación (MC)*

El objetivo principal de este módulo es proveer un *marco común de comunicación* entre el *MCER y los agentes de ejecución.* Independientemente del tipo de acción de respuesta a ejecutar sobre el *agente de ejecución* éste módulo debe proveer servicios de entrega confiable y segura de las solicitudes *de acción de respuestas activas y pasivas* inferidas por el razonador AIRS y construidas por el MCER.

- **Comunicación confiable**: Puesto que la comunicación entre el *MCER* y el *agente de ejecución* debe ser confiable, el establecimiento de conexión se efectúa por medio de protocolo de transporte TCP. El *agente de ejecución* escucha a través de un puerto que es conocido de antemano por el MCER.
- **Comunicación cifrada:**Las solicitudes de acción de respuesta que viajan a través de la red se cifran utilizando el algoritmo de cifrado en bloques simétrico TwoFish. El tamaño del bloque en TwoFish es de 128 bits y el tamaño de la clave puede alcanzar los 256 bits. Éste algoritmo de cifrado es muy robusto y alcanzó la ronda final del concurso del NIST para reemplazar el algoritmo DES [56].

El *marco común de comunicación* permite el despliegue de varias acciones de respuesta sobre componentes de seguridad remotos en los cuales se ha instalado un agente de ejecución. Cabe recalcar que éste módulo reutiliza lo implementado por la herramienta SnortSam, abordado en la sección 5.1. Se proponen algunos cambios y adaptaciones para que pueda operar con el AIRS basado en ontologías. Para ello se ha de especificar un protocolo de comunicación entre el MCER y los agentes de ejecución teniendo en cuenta las siguientes consideraciones:

- Se define un *formato de paquete* de solicitud que incluya todos los parámetros necesarios tanto para el establecimiento y cifrado de la comunicación, como para la ejecución de la respuesta.




Los campos *SourceIP, DestinationIP, Duration, SrcPrt, DstPort, Protocol, SID Mode, Plugin y User* contienen la información que será usada por los agentes de ejecución y su respectivo plugin para le ejecución de una acción de respuesta. Estos campos se abordan con más detalle en el capítulo 6.

Los campos *agentSeqNo y airsSeqNo* son utilizados para intercambiar números de secuencia necesarios en el proceso de cifrado y descrifrado.

- Se define un conjunto de mensajes a intercambiar entre el MCER y los agentes de ejecución. El tipo de mensaje está determinado por el campo *Status* del formato de paquete establecido anteriormente. SnortSam define los siguientes tipos de mensajes:
    - Desde el MCER hacia el agente de ejecución:
        - **Checkin (status=1)**: Es el primer mensaje que envía el MCER hacia el *agente de ejecución* y su objetivo es intercambiar el modificador de clave y números de secuencia utilizados para cifrar la información.
        - **Block (status=3):** Es la solicitud de acción de respuesta.
        - **Unblock (status=4):** Es una solicitud para deshacer una acción de respuesta.
        - **Checkout (status=2)**: Es una solicitud de desconexión, además en el agente de ejecución se eliminan todas las tablas asociadas al MCER que envió dicha solicitud.
    - Desde el agente de ejecución hacie el MCER:
        - **Ok (status=4):** Indica el éxito de la ejecución de una respuesta. Es importante indicar, que este mensaje no indica que la respuesta haya tenido efecto, únicamente valora que se pudo ejecutar la respuesta.
        - **Error (status=5):** Es un mensaje que indica que la acción de respuesta no pudo ser ejecutada.
        - **NewKey (status=6):** El agente de ejecución solicita al MCER la definición de una nueva clave de cifrado.
        - **Resynk (status=7):** El agente de ejecución solicita una resincronización de los números de secuencia y modificadores de contraseña.
        - **Hold(status=8):** El agente de ejecución solicita que el MCER no cierre la conexión luego de ejecutar una acción de respuesta.

### *Agente de Ejecución*

El *agente de ejecución* actúa como un servicio que siempre se está ejecutando a la espera de la llegada de nuevos paquetes de solicitud de respuesta. Ante el arribo de un paquete de solicitud, selecciona el plugin adecuado y ejecuta el conjunto de comandos asociados a una *respuesta activa o pasiva* sobre un *componente de seguridad*.

La primera tarea que lleva a cabo el agente de ejecución es la inicialización de plugins, los mismos que han sido registrados por el gestor de plugins y que se tratan en mayor detalle en la siguiente sección.

Luego el agente de ejecución entra en un proceso de espera de paquetes de solicitud de respuesta. Cuando el *agente de ejecución* recibe un paquete de solicitud de respuesta a través del *módulo de comunicación,* primero verifica la autenticidad del emisor de la solicitud de acción de respuesta. Si el emisor es auténtico, entonces se descifra el mensaje y se verifica que la solicitud no esté dirigida hacia uno de los componentes que se encuentra en *lista blanca.*

Finalmente el *agente de ejecución* selecciona el plugin especificado en el paquete de solicitud y llama a la rutina respectiva para la ejecución de la acción de respuesta sobre el componente de seguridad respectivo. Al término de la ejecución el agente envía de vuelta un resultado que es almacenado en un archivo de logs del *MCER*.

Otra de las tareas que lleva a cabo el *agente de ejecución* es el control del tiempo que tendrá efecto una acción de respuesta, tal como se mencionó anteriormente, el MCER define un valor de tiempo que tendrá efecto una respuesta. Este valor es almacenado por el *agente de ejecución* sirve para deshacer una acción de respuesta.

### Gestor de Plugins

El *gestor de plugins* es un componente importante dentro de la arquitectura del ejecutor de respuestas ya que permite resolver el problema de la ***heterogeneidad*** de los *componentes de seguridad* con los que tiene que interactuar el *agente de ejecución*. Una solicitud de respuesta siempre tiene la misma estructura, independientemente del *componente de seguridad*; no obstante, cada *componente de seguridad* tiene su propia sintaxis e interfaz de línea de comandos, y requiere parámetros específicos para su ejecución. Mediante el encapsulamiento de la lógica de control de una *acción de respuesta* en un plugin, el ejecutor de respuestas puede ser utilizado como una *plataforma escalable* para desplegar nuevas acciones de respuesta sobre diversos *componentes de seguridad* sin tener la necesidad de modificar otros módulos del ejecutor de respuestas.

El gestor de plugins permite el manejo de varios *componentes de seguridad* a través del despliegue de plugins, con el objetivo de resolver algunos problemas que incluyen básicamente 3 aspectos:

- Cada componente de seguridad requiere de diferentes parámetros de configuración: iptables, requiere de la interfaz de red sobre la que se aplicará una regla; honeynet VNX, requiere el nombre de usuario y la ruta del archivo VNX; router cisco, requiere dirección IP y contraseña. Entonces

el *plugin* tiene que separar cualquier dependencia de la dotación de los parámetros del *agente de ejecución.*

- Cada componente de seguridad tiene sus propios comandos de configuración y requiere una lógica de control específica de dicho componente; por consiguiente, el plugin tiene que encapsular la lógica de control.
- Cada componente de seguridad tiene diferentes capacidades en cuanto a la posibilidad de deshacer una acción, soportar múltiples hilos de ejecución, requerir una función de keepalive, entre otros. Esta información es requerida por el agente de ejecución y por tanto debe ser proporcionada en el registro del plugin.

Como se observa en la Figura 21, el gestor de plugins provee dos interfaces: una *interfaz de registro* de plugins y una *interfaz de consulta* de plugins.

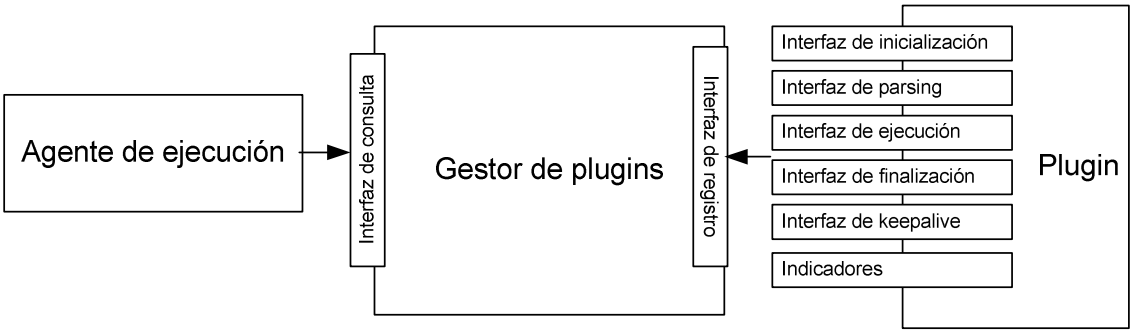



**Figura 21. Interfaces del gestor de plugins.**

La *interfaz de registro* permite el registro de un nuevo plugin, para lo cual el plugin ha de proveer de cinco interfaces y ciertos indicadores necesarios para la operación del *agente de ejecución*.

**Interfaz de inicialización:** Utilizado para ejecutar una rutina de inicio que verifique ciertas precondiciones requeridas para ejecutar una respuesta. Si la rutina de inicio no se ejecuta correctamente, entonces el plugin es deshabilitado. Por ejemplo, la *respuesta activa de recuperación* que restaura los archivos de un sitio web, requiere de un cliente FTP que se conecta al servidor para descargar ciertos archivos. Entonces se puede comprobar previamente que el agente de ejecución cuente con un cliente FTP para acceder al servidor, caso contrario el plugin es deshabilitado.

**Interfaz de parsing:** Utilizado para ejecutar una rutina de parsing de los parámetros de configuración requeridos para ejecutar una respuesta sobre un componente de seguridad. Por ejemplo, *la respuesta activa de protección* que agrega una lista de control

de acceso a un router perimetral cisco, requiere: la dirección IP del router y las contraseñas respectivas.

**Interfaz de ejecución:** Utilizado para hacer referencia a la rutina que contiene la lógica de control que lleva a cabo la acción de respuesta. Por ejemplo, *la respuesta activa de decepción* que despliega una red señuelo usando VNX, incluirá todos los comandos propios de VNX que permitan la ejecución de dicha respuesta.

**Interfaz de finalización:** Utilizado para ejecutar una rutina de salida en el momento en que el ejecutor de respuestas se cierra. Esto le otorga al plugin la posibilidad de limpiar o ejecutar acciones a conveniencia.

**Interfaz de keepalive:** Utilizado para ejecutar una rutina de keep-alive, cuya función es mantener conexiones persistentes con componentes de seguridad que así lo requieran. Por ejemplo puede ser necesario mantener una conexión con un firewall perimetral para evitar realizar logins frecuentes.

**Indicadores:** Varios indicadores pueden ser requeridos durante la fase de registro del plugin y pueden ser añadidos conforme a las necesidades del agente de ejecución. Por ejemplo una función del agente de ejecución es deshacer una acción de respuesta después de un tiempo determinado; no obstante, algunas respuestas como por ejemplo la *restauración de ficheros,* no requieren esta función; por lo tanto a través de un indicador se dice al agente de ejecución que dicho plugin no requiere esa funcionalidad.

#### *Componentes de seguridad*

Representa el dispositivo que lleva a cabo la acción de respuesta real utilizando su interfaz de línea de comandos, tal que se altere su funcionamiento tan pronto como es ejecutado. Un router, firewall, servidor web, servidor FTP, gestor de usuarios, gestor de procesos son algunos ejemplos de ellos.

# 6 Implementación

## 6.1 Entorno de desarrollo

El requisito no funcional NF01, definido en la Tabla 20, establece que el *razonador AIRS* está desarrollado en Java; por lo tanto el MCER al formar parte de éste, igualmente es desarrollado utilizando el lenguaje de programación Java.

Por otro lado, ya que la arquitectura propuesta reutiliza ciertos componentes de SnortSam que serán adaptados a la arquitectura propuesta, se emplea el lenguaje ANSI C para el *módulo de comunicación, el agente de ejecución y el gestor de plugins*.

En la Tabla 24 se muestra información detallada del entorno de desarrollo empleado.

| | Entorno de desarrollo |
|---|---|
| Sistema operativo | GNU Linux Ubuntu 12.04.1 LTS |
| Lenguaje de programación | *Razonador AIRS y MCER:*<br>• Java 1.6.04_24 OpenJDK 64-Bits.<br>• IDE Netbeans 7.0.1 |
| | *Módulo de comunicación, agente de ejecución y gestor de plugins:*<br>• ANSI C<br>• Compilador GCC 4.6.3 |
| Máquina | Dell XPS L502X |

**Tabla 24. Entorno de desarrollo del ejecutor de respuestas..**

## 6.2 IDSs

La arquitectura propuesta trabaja con dos IDSs: el IDS basado en red Snort y el IDS basado en host OSSEC.

### 6.2.1 Snort

*Snort* [57] es un NIDS open source que combina los beneficios de los dos métodos de detección actualmente empleados: el método de detección basado en anomalías y el basado en usos indebidos; ambos métodos fueron revisados en la sección 2.1.2. La fuente de datos de Snort son los paquetes que circulan a través de una interfaz de red y que pertenecen a los hosts que se requiere proteger. Un IDS, en la medida de lo posible, no debe interferir con el funcionamiento normal de la red; es decir que su operación no debería introducir retardos en la entrega de paquetes o consumir recursos computacionales y de memoria excesivos. En virtud de ello, los NIDS Snort funcionarán sobre hosts independientes y se empleará la *TAP Daemonlogger*[20],

[20] http://www.snort.org/snort-downloads/additional-downloads

desarrollada por Martin Roesch, que funciona como sniffer sobre una interfaz y reescribe los paquetes hacia una segunda interfaz, a la cual está conectado el IDS. Por ejemplo en la Figura 22, el *daemonlogger* instalado sobre el host R-FW reescribe los paquetes que ingresan por la interfaz NetATT hacia la interfaz IDSNet1 a la cual se encuentra conectado el IDS-1 Snort.

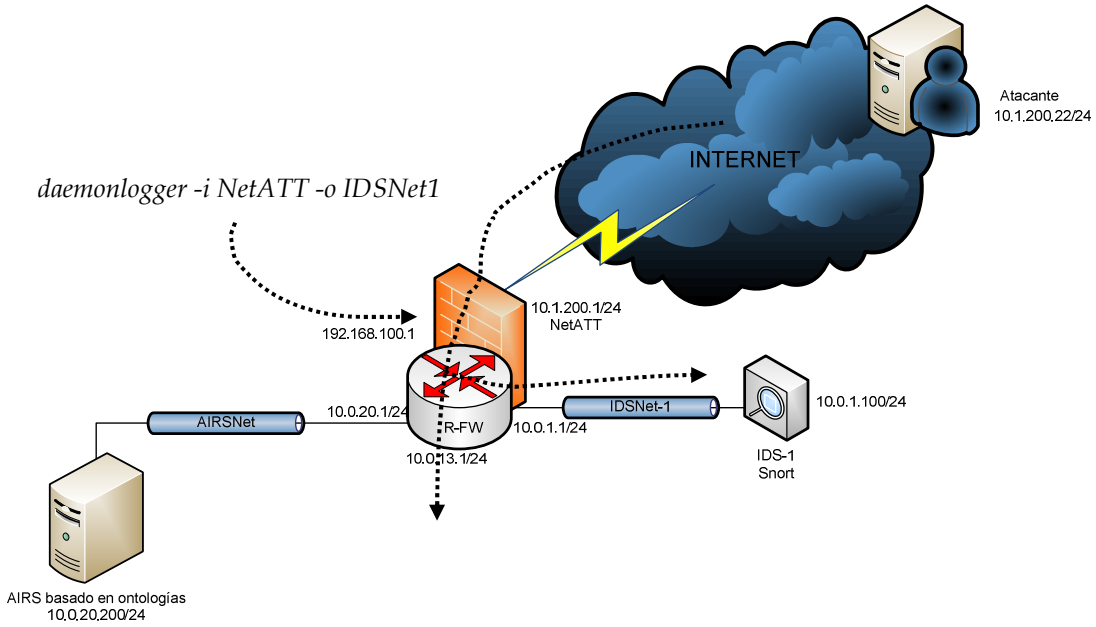



**Figura 22. Uso de daemonlogger para reescribir los paquetes desde la interfaz NetATT hacia IDSNet1.**

Los paquetes reescritos por *daemonlogger* son recibidos por el IDS Snort y ante la detección de una intrusión se genera una alerta en un formato requerido. En nuestro caso, las alertas son enviadas en formato syslog {{231 Gerhards, Rainer 2009}} hacia el servidor *syslogd*, que escucha a través del puerto por defecto 514. Esto requiere la edición de dos archivos de configuración.

- En el fichero de Snort *snort.conf* se define que el envío de alertas se realizará hacia el servidor *syslogd:*

*output alert_syslog: LOG_AUTH LOG_ALERT*

Las alertas emitidas hacia el servidor *syslogd* serán recibidas por el módulo *receptor de alertas* del AIRS basado en ontologías.

### 6.2.2 OSSEC

OSSEC es un HIDS open source multiplataforma que básicamente cumple con tres tareas: verificación de integridad de ficheros, monitorización y análisis de logs del sistema, y detección de rootkits [58]. La fuente de datos de OSSEC son los ficheros y logs que requieren ser monitorizados y que han sido especificados por el

administrador en el archivo de configuración. Ante la detección de una intrusión, el servidor OSSEC envía una alerta en formato syslog hacia el puerto 514 del servidor *syslogd,* en donde también se encuentra el AIRS basado en ontologías que recepta las dichas alertas a través de su módulo *receptor de alertas*. Se requiere la configuración del archivo *ossec.conf,* en el que se debe especificar:

- La dirección del servidor *syslogd:*

```
<syslog_output>
        <server>10.0.1.1</server>
        <port>514</port>
</syslog_output>
```

- La habilitación del cliente syslog de OSSEC:

*/var/ossec/bin/ossec-control enable client-syslog*

Las alertas emitidas hacia el servidor *syslogd* serán recibidas por el módulo *receptor de alertas* del AIRS basado en ontologías.

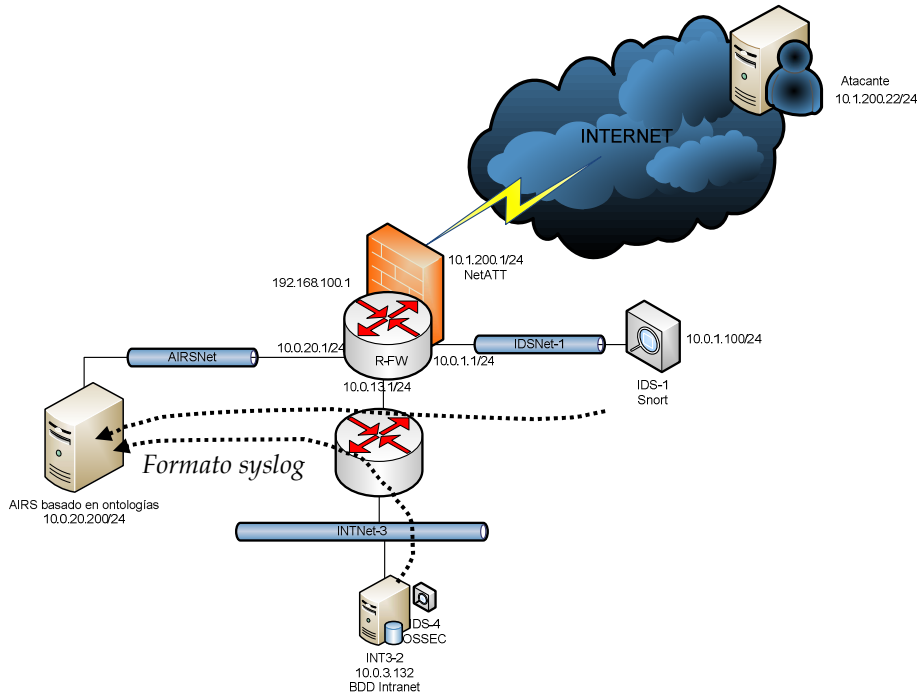



**Figura 23. Envío de alertas desde los IDSs hacia el AIRS basado en ontologías.**

Por supuesto la configuración de los IDSs va más allá de lo explicado en este apartado, la definición de reglas, habilitación de preprocesadores, definición de archivos a monitorizar, entre otras tareas, deben ser llevadas a cabo y dependerá de las necesidades específicas de cada organización; lo esencial es comprender que los IDSs monitorizan diferentes fuentes de datos y notifican la detección de una intrusión al

AIRS basado en ontologías, mediante el envío de una alerta en formato syslog. Ver Figura 23.

## 6.3 Razonador AIRS

El razonador del AIRS basado en ontologías se encuentra en el mismo host que el servidor *syslogd*. El funcionamiento detallado del AIRS fue realizado en la sección 2.4, y para nuestro caso será considerado como una *caja negra* que cumple con dos funciones:

- A través del puerto 512 el módulo de *recepción de alertas*, recibe las alertas en formato syslog que son generadas por los IDSs Snort y OSSEC, y parsea a sus correspondientes conceptos de la ontología. Dado que las alertas son emitidas hacia el puerto 514 (puerto por defecto de syslogd) se añade una regla a la cadena PREROUTING de la tabla NAT del firewall IPTables para redirigirlo hacia el *receptor de alertas* del AIRS:

  *iptables -A PREROUTING -t nat -p udp --dport 514 -j DNAT --to-destination 10.0.1.1:512*
- Infiere la respuesta óptima en base las métricas de respuesta de *reducción de daño y mínimo costo o reducción de daño y alta severidad y eficiencia* y llama al módulo *ejecutor de respuestas* para hacer efectiva una acción de respuesta sobre un componente de seguridad.

Como se puede constatar en la Figura 25, el componente del *ejecutor de respuestas* que interactúa con el *razonador AIRS* es el *módulo central de ejecución de respuestas (MCER)*. Ambos componentes han sido desarrollados en Java y para permitir una comunicación entre ellos se ha definido una interfaz de comunicación a la que se denomina *Razonador-MCER;* por consiguiente, cada vez que el razonador infiera una respuesta, llamará al *módulo central de comunicación* de la siguiente manera:

*CentralModuleExecution(String inferredResponseName, ResponseActionParams paramsAlert)*

- La cadena *inferedResponseName* es el nombre de la respuesta inferida y corresponde con el valor de la propiedad *responseAction* de la ontología del AIRS.
- Una instancia de la clase *ResponseActionParams* (Figura 24) que contiene los parámetros relacionados a la intrusión y que serán utilizados por el MCER para ejecutar una respuesta.

```java
/**
 * Clase que contiene los parámetros que son emitidos por el Razonador AIRS al MCER
 * @param _hids:Indica si la alerta proviene de un HIDS (true) o un NIDS (false)
 * @param _sid:Identificador de alerta de intrusión.
 * @param _intrusionType:Tipo de intrusión clasificado de acuerdo a una taxonomía.
 * @param _mainIP:Dirección IP del host atacante.
 * @param _peerIP:Dirección IP de host atacado.
 * @param _protocol:Protocolo empleado en el ataque.
 * @param _portConn:Puerto del host atacado.
 * @param _user:Usuario con el que se intenta perpetrar un ataque.
 * @param _adParam:Parámetro adicional.
 */
public class ResponseActionParams {
    private Boolean _hids;
    private String _sid;
    private String _intrusionType;
    private String _mainIP;
    private String _peerIP;
    private String _protocol;
    private Integer _portConn;
    private String _user;
    private String _adParam;
```

**Figura 24. Atributos de la clase ResponseActionParams..**

De esta manera, cada vez que el r*azonador AIRS* infiere una respuesta debe instanciar previamente un objeto de la clase *ResponseActionParams,* asignando valores a los atributos necesarios para ejecutar una respuesta. Los valores a proveer dependen del tipo de respuesta a ejecutar; por supuesto el razonador AIRS conoce de antemano que parámetros son requeridos para cada respuesta y los obtiene ya sea de la alerta de intrusión o empleando herramientas de gestión remota como Nagios y Net-SNMP, por mencionar algunas.

## 6.4 Módulo central de ejecución de respuestas

Una vez que el *razonador AIRS* infiere la respuesta óptima, entra en escena el *MCER* quien recibe una cadena con el nombre de la respuesta inferida y los parámetros requeridos para su ejecución, enviadas dentro de un objeto de la clase *ResponseActionParams* a través de la interfaz *MCER-Razonador* definida en la sección anterior. En la Figura 25 se muestra el diagrama de despliegue del MCER, la implementación de este módulo se ha dividido en dos partes:

1. Definición y parseo de los parámetros fijos que deben ser provistos por el administrador, en los ficheros de configuración (*airsResponseExecutor.conf y plugin-numbers.conf)* y que coincida con el modelo de datos establecido en la Figura 20.
2. Implementación de la lógica de control para seleccionar los parámetros necesarios para ejecutar una respuesta y que serán enviados posteriormente al módulo de comunicaciones a través de la interfaz *Communication-MCER.*

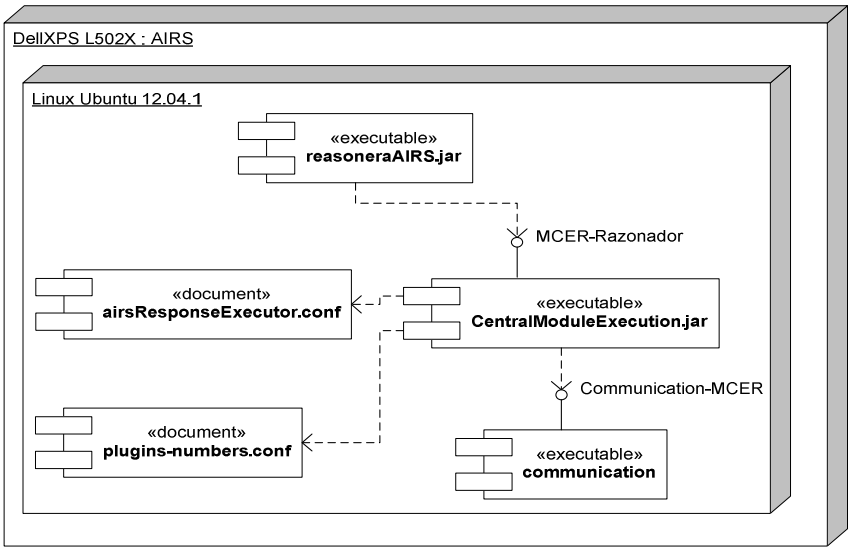



Figura 25. Diagrama de despliegue del módulo central de ejecución de respuestas.

### 6.4.1 Definición y parsing de parámetros fijos de las acciones de respuesta

La primera tarea llevada a cabo por el MCER consiste en el parseo de algunos parámetros fijos provistos por el administrador, que entre otras cosas, permiten la localización de los *agentes de ejecución* (el modelo de datos fue definido en el diagrama de la Figura 20).

Los parámetros fijos son especificados por el administrador en el fichero de configuración *airsResponseExecutor.conf*. Un ejemplo de configuración se muestra en la Figura 26, y proporciona la siguiente información:

- En la *línea 8* se especifica el nombre del fichero en donde se registran los logs de ciertos eventos. Estos podrían ser solicitudes acciones de respuesta enviadas al módulo de comunicaciones o errores producidos. Si únicamente se define el nombre del fichero se almacenará en la ruta */var/log/*. Además la sentencia definida en *línea 11* complementa dicha funcionalidad al concretar el nivel de registro requerido:
  - 0: No se realizan registros.
  - 1: Se registran únicamente los errores.
  - 2: Se registran los errores y las solicitudes de respuesta enviadas hacia el módulo de comunicaciones.
  - 3: Se registran los errores, solicitudes de respuesta e información adicional.

  Esta funcionalidad da cumplimiento al requisito no funcional de mantenibilidad NF07 definido en la Tabla 20.

- De la línea 14 a la 18 se puntualizan los *agentes de ejecución* sobre los que se requiere ejecutar una acción de respuesta. Por cada agente se debe definir un nombre o identificador, la dirección IP, el puerto y la contraseña de cifrado.
- De la línea 21 a la 24 se definen grupos de *agentes de ejecución* a los que se emiten las solicitudes de acción de respuesta. Por ejemplo en la línea 23, el grupo *IntranetCriticalHostGroup* está constituido por los agentes de ejecución *INT3-1 e INT3-2*, que son dos servidores importantes de la intranet de la organización.
- Aunque es una configuración opcional, en las líneas 27 y 28 se especifican grupos de identificadores de intrusiones ante los cuales se enviará una solicitud de respuesta. Tanto Snort como OSSEC manejan identificadores de alertas. Por ejemplo *DoSGroup* agrupa tres intrusiones específicas, identificadas por su SID: 636, 678 y 564. Cuando el *MCER* recibe una alerta, cuyo SID coincide con alguno de ellos, entonces ejecuta la acción de respuesta que lo referencia; por ejemplo la acción de respuesta *honeyneyDeployment* configurada en la *línea 40*.

```
1  ##########################################################################################################
2  # CONFIGURATION FILE
3  # 2012 Danny Guamán <ds.guaman@alumnos.upm.es>
4  # Universidad Politécnica de Madrid
5  ##########################################################################################################
6  #
7  # logfile: <filename>
8  logfile: airsResponseExecutor.log
9  #
10 # loglevel: <level>
11 loglevel: 2
12 #
13 # executor_agent: <nameExecutor> <AIRS Executor Agent>:<port>/<key>
14 executor_agent: R-FW, 10.0.20.1:34000, mlp455w0rd
15 executor_agent: local-agent, 127.0.0.1:898, mlp455w0rd
16 executor_agent: INT3-1, 10.0.3.131, mlp455w0rd
17 executor_agent: INT3-1, 10.0.3.132, mlp455w0rd
18 executor_agent: HN-Server, 10.0.12.100, mlp455w0rd
19 # group: <nameGroup>/<nameExecutors(s)>
20 #
21 group: PerimetralFirewallGroup > R-FW
22 group: LocalAgentGroup > local-agent
23 group: IntranetCriticalHostGroup > INT3-1, INT3-2
24 group: HoneyNetServerGroup > HN-Server
25 #
26 # sid_group: <nameSidGroup> > <number SIDS..>
27 sid_group:BackdoorGroup > 32456,12343,12343,45633,45677
28 sid_group:DoSGroup > 636, 678, 564
29 #
30 # response_action: <Name>, <executors group>, <plugin>, <who>, <execute/undo>, <duration>, <mode-block>, <sid-group>
31 response_action: blockInAttack, PerimetralFirewallGroup, ipt-block, src, execute, 60, out, ALL
32 response_action: blockOutAttack, PerimetralFirewallGroup, ipt-block, src, execute, 60, out, ALL
33 response_action: blockInOutAttack, PerimetralFirewallGroup, ipt-block, src, execute, 60, both, ALL
34 response_action: blockThisAttack, PerimetralFirewallGroup, ipt-block, src, execute, 60, this, ALL
35 response_action: mailNotification, LocalAgentGroup, email, 0, execute, 0, 0,ALL
36 response_action: addACL, LocalAgentGroup, ciscoacl, src, execute, 24hours ,conn, ALL
37 response_action: denyConnectionHost, IntranetCriticalHostGroup, host-deny, src, execute, 1hour, in, ALL
38 response_action: disableLinuxUser, THIS, disable-user, dst, execute, 1hour, 0, BackdoorGroup
39 response_action: redirectTraffic, PerimetralFirewallGroup, ipt-redirect, src, execute, 2hours, conn, DoSGroup
40 response_action: honeynetDeployment, HoneyNetServerGroup, hnvnx, 0, execute, 2hours, 0, DoSGroup
41 #
42 # composed: <response_action_name_1>   >    <response_action_name_2><reponse_action_name_3>....<Sid-Group>
43 composed:relayAttack  >  redirectTraffic, honeynetDeployment, mailNotification:ALL
```

**Figura 26. Ejemplo del archivo de configuración airsResponseExecutor.conf.**

- De la línea 31 a la 40 se definen las acciones de respuestas y los parámetros fijos asociados a ellas:
    - *Name:* Nombre de la acción de respuesta. Debe coincidir con el valor de la propiedad *actionResponse* de la ontología.
    - *Executors Group:* Agentes de ejecución sobre los que se ejecuta la respuesta. En ocasiones será necesario actuar sobre el host que está siendo atacado, en este caso una asignación fija no es útil, ya que solo se podrá identificar sobre qué host actuar el momento que se detecta una intrusión. Si éste fuese el caso se debe especificar la cadena THIS, entonces el MCER obtiene la dirección del *agente de ejecución* sobre la que actuará de los parámetros proveídos por el razonador AIRS. Por ejemplo la acción de respuesta de la línea 38 define la cadena THIS, ya que tiene por objetivo deshabilitar un usuario sobre el host que se está perpetrando el ataque.
    - *Plugin:* Nombre del plugin conectado al *agente de ejecución* que ejecutará la acción de respuesta sobre el componente de seguridad respectivo. Existe un archivo de configuración *plugin-numbers.conf,* en donde constan todos los plugins implementados y un número que lo identifica (Ver Figura 27). Dicho número será enviado en el paquete de solicitud de respuesta. En la sección 6.5 se aborda el tema concerniente al paquete de solicitud.
    - *Who:* Identifica sobre que host actuará la respuesta: sobre el atacante (src) o sobre la víctima (dst). Por ejemplo para la respuesta de la línea 31, que consiste en el bloqueo del tráfico de entrada sobre un firewall perimetral IPTables, se debe especificar que se actuará sobre el atacante mediante la cadena src; ya que el objetivo es filtrar a dicho host atacante y no a la víctima. Ahora, en la línea 38, el objetivo de la respuesta es bloquear al usuario que se encuentra en el host de la víctima, en este caso se debe especificar la palabra dst.
    - *Execute/undo:* Puesto que el agente de ejecución tiene la funcionalidad de ejecutar una acción de respuesta, cuyo efecto tendrá una duración específica; esta opción permite deshacer una acción de respuesta antes de que dicho intervalo de tiempo se cumpla.
    - *Duration:* Intervalo de tiempo que tendrá efecto una acción de respuesta, se puede expresar en segundos, minutos, horas o días.
    - *Mode-Block:* Esta opción se hereda de la herramienta SnortSam y determina el criterio en base al cual se ejecuta una acción de respuesta basada en red. En función del tráfico de entrada (in), salida (out), entrada y salida (inout) o de esa conexión específica (conn).

- *Sid-Group:* Grupo de identificadores de intrusiones ante los cuales se ejecuta la acción de respuesta. Si se requiere ejecutar la acción de respuesta independientemente del SID se debe definir la cadena ALL.

```
3  # <PluginHandler>:<number>
4  #   <PluginHandler>: Nombre del plugin.
5  #   <number>: Número identificador entre 2-65535.
6  #             Importante: "1" es reservado para todos "ALL" los plugins.
7  ipt-block:2
8  hnvnx:3
9  email:4
10 email-execution-only:5
11 ciscoacl:6
12 ipt-redirect:7
13 disable-user:8
14 backup-mysql:9
15 restore-mysql:10
16 host-deny:11
17 host-null-route:12
18 cisconullroute:13
19 whitelist:14
20 upload-backup-ftp:15
21 download-backup-ftp:16
22 close-port-linux:17
23 close-connection-linux:18
```

**Figura 27. Asignación de identificador a los plugins registrados sobre los agentes de ejecución en el archivo plugin-numbers.conf.**

- Finalmente la línea 43 define una respuesta compuesta por varias acciones anteriormente definidas. En este caso la respuesta compuesta *relayAttack* ejecuta tres acciones: redirige el tráfico hacia un host en donde posteriormente se despliega una honeynet y finalmente envía un correo electrónico hacia el administrador.

Para parsear la información de los archivos de configuración antes mencionados se ha implementado la clase *FixedParamsFormatter,* cuyo diagrama de secuencia se muestra en la Figura 28. Además por cada tabla definida en el modelo de datos de la Figura 20 se ha implementado una clase con sus respectivos atributos; la representación en objetos facilita la implementación de la lógica de control del *MCER* que se explica en la siguiente sección.

Cómo se observa en el diagrama de despliegue de la Figura 28, la instancia *fixedParams* de la clase *FixedParamsFormatter* es llamada cuando arranca el *razonador AIRS,* y ejecuta básicamente dos funciones:

- *ParsePlugin:* Realiza el parsing del fichero *plugin-numbers.conf,* en donde se especifican los números de plugins que se encuentran conectados a los agentes de ejecución.
- *ParseConfig:* Realiza el parsing del fichero *airsResponseExecutor.conf.* Los parámetros incluidos en este archivo fueron abordados anteriormente. Cinco funciones han sido implementadas para parsear cada línea de configuración a su respectiva representación en objetos: *ParseExecutorAgent,*

*ParseResponseAction, ParseGroupExecutorAgent, ParseSidsGroup y ParseComposite.*

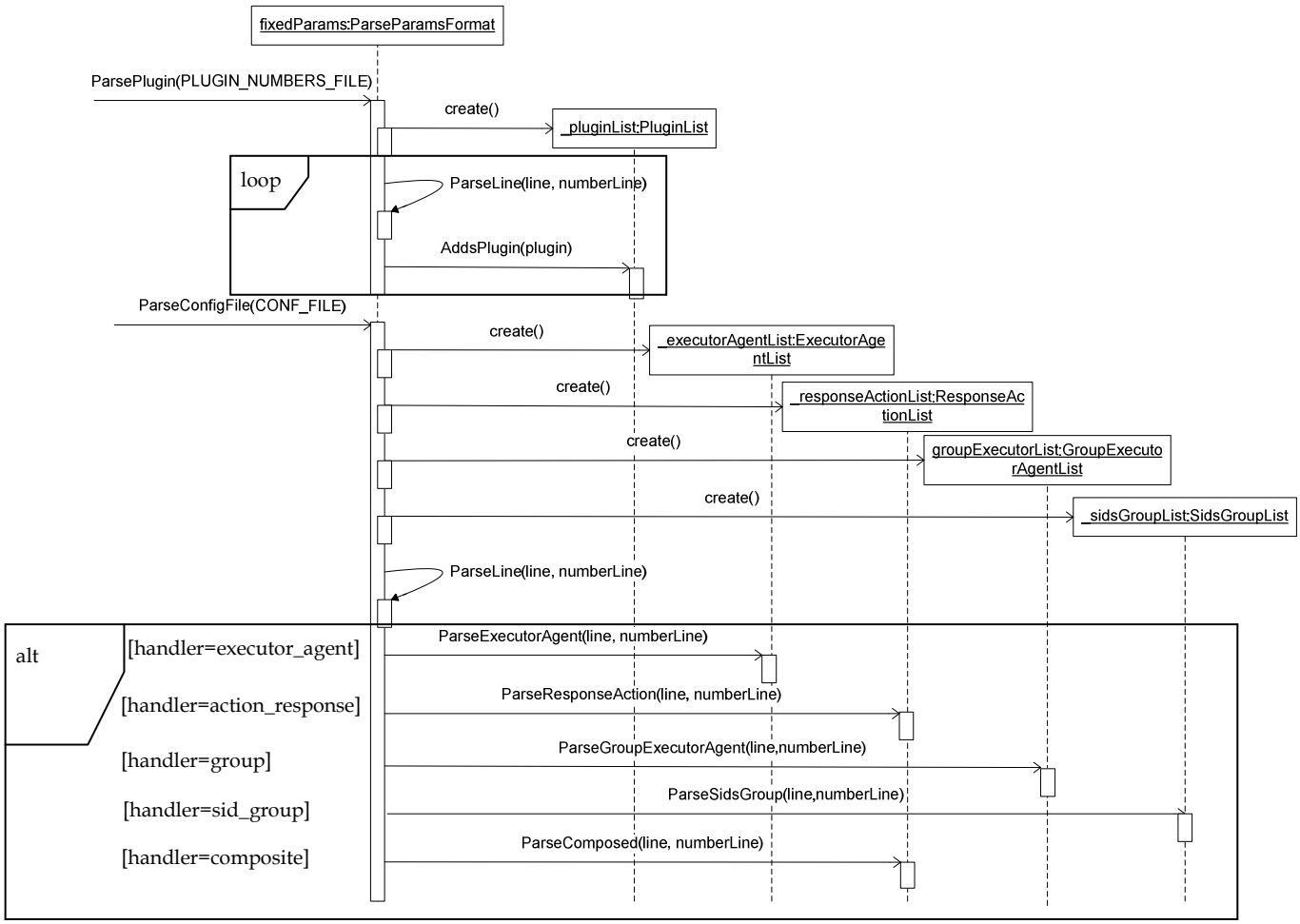



**Figura 28. Diagrama de secuencia del objeto fixedParams de la clase FixedParamsFormatter.**

Ambas funciones, *ParsePlugin y ParseConfig*, cargan toda la información definida por el administrador en objetos de tipo ArrayList y están disponibles, junto con los parámetros provistos a través de la interfaz MCER-Razonador, para ser utilizadas en la lógica de control para la selección de los parámetros necesarios para emitir una solicitud de acción de respuesta hacia el módulo de comunicación y posteriormente hacia el agente de ejecución.

### 6.4.2 Lógica de control para selección de parámetros

Luego de que el *razonador AIRS* infiere la respuesta óptima a través de la función *inferOptimumResponse,* crea una instancia de la clase *CentralModuleExecution* proporcionándole el nombre de la respuesta inferida y los parámetros requeridos por esa respuesta. Posteriormente llama al procedimiento *BuildResponseActionRequest().*

```
CentralModuleExecution MCER;
MCER = new CentralModuleExecution(inferredResponse, params);
MCER.BuildResponseActionRequest();
```

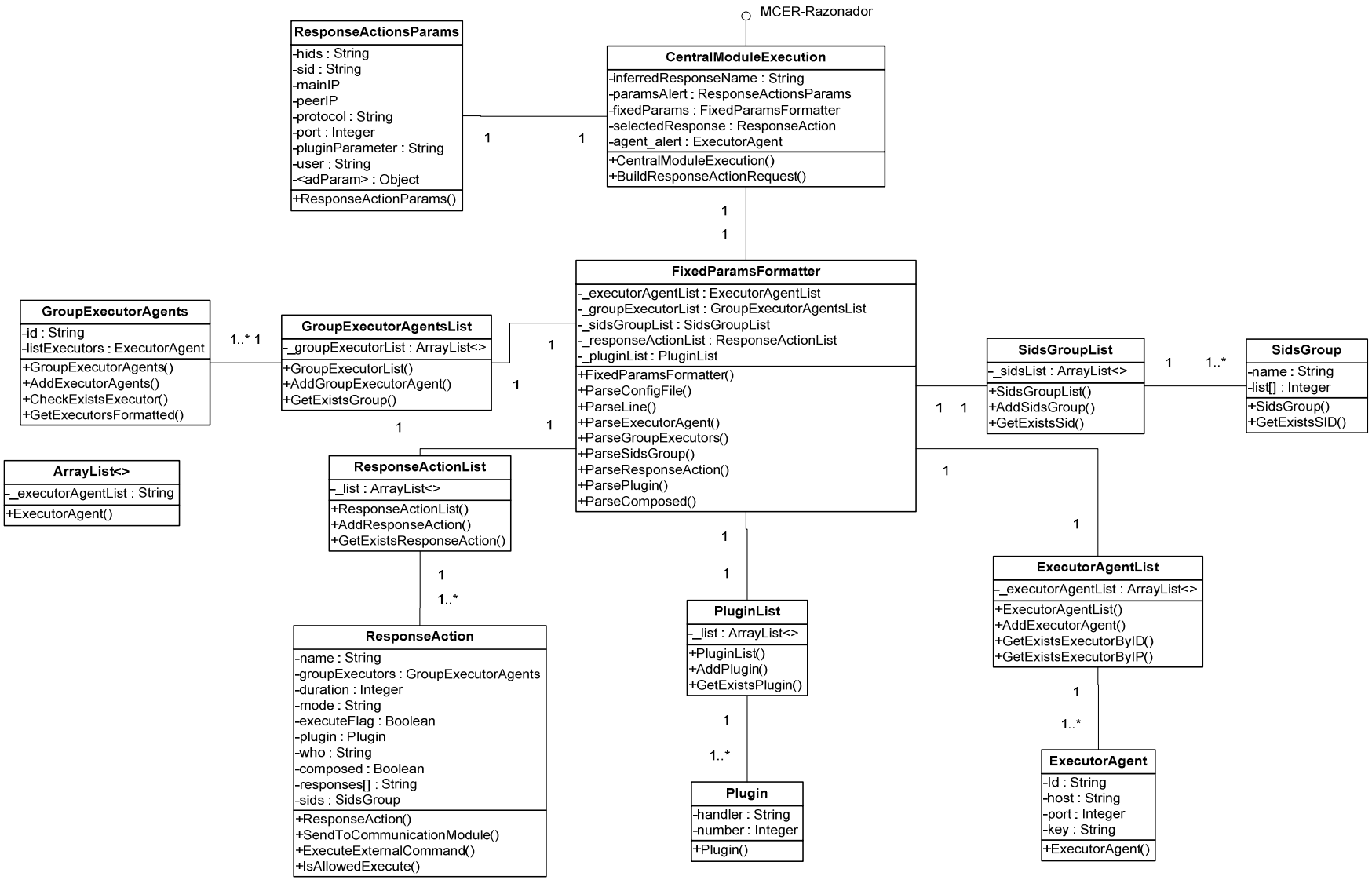



Figura 29. Diagrama de clases del módulo central de ejecución de respuestas.

En la Figura 30 se muestra la secuencia de operación que ha sido implementada en el *módulo central de ejecución de respuestas*. Básicamente lleva a cabo las siguientes tareas:

1. Se obtiene toda la información proporcionada por el administrador y que ha sido parseada en el arraque del AIRS. Se invoca la función *getFixedParams()*.
2. En base al nombre de la respuesta inferida *inferredResponseName* proporcionada por el *razonador AIRS,* se obtienen todos los parámetros fijos asociados a dicha respuesta (ver sección 6.4.1). Para ello se llama a la función *getExistsResponseAction(inferredResponseName)*.
3. Una vez obtenida la respuesta seleccionada *selected_response,* mediante la llamada a la función *getComposed()* sobre el objeto *selected_response,* se verifica si la respuesta a ejecutar es simple o compuesta.
    a. Si la respuesta es simple (*boolean==false*), entonces las tareas 4, 5 y 6 se ejecutan por una sola vez.
    b. Si la respuesta es compuesta (*boolean=true*), entonces las tareas 4, 5 y 6 se ejecutan un número de veces igual al número de acciones que componen dicha respuesta compuesta.
4. Sobre la respuesta seleccionada *selected_response* se obtiene el grupo de agentes de ejecución y se *verifica el perfil* asignado:
    a. Si el perfil (ID) asignado es *THIS,* entonces se obtiene la dirección IP (*mainIP)* contenida en el objeto *paramsAlert* que se recibe desde desde el razonador. Posteriormente se busca un *agente de ejecución* que coincida con dicha dirección IP:
        i. Si se encuentra un agente, entonces se obtiene los datos restantes de dicho agente: puerto de conexión y contraseña.
        ii. Si no se encuentra el agente, entonces se utiliza se utiliza el puerto de conexión por defecto (898) y contraseña por defecto.
    b. Si el perfil (ID) es diferente de *THIS,* entonces no es necesario realizar la búsqueda anterior, ya que los datos han sido asignados por el administrador.
5. Sobre la respuesta seleccionada, se llama a la función *SendToCommunicationModule,* que envía una solicitud de respuesta hacia el *módulo de comunicación* a través de la interfaz *Communication-MCER*. Para ello se lleva a cabo las siguientes tareas:
    a. Se crea el objeto *externalCmd* de la clase *Process,* quien llama al *módulo de comunicación*, que está desarrollado en ANSI C, que se encarga de la transmisión de la solicitud a través de la red.

```
cmd = "./communication-debug -vv -e " + cmd;
externalCmd = Runtime.getRuntime().exec(cmd);
```

    b. Previo a la llamada del módulo de comunicación, se contruye una solicitud (*cmd*), únicamente con los parámetros que no son nulos.

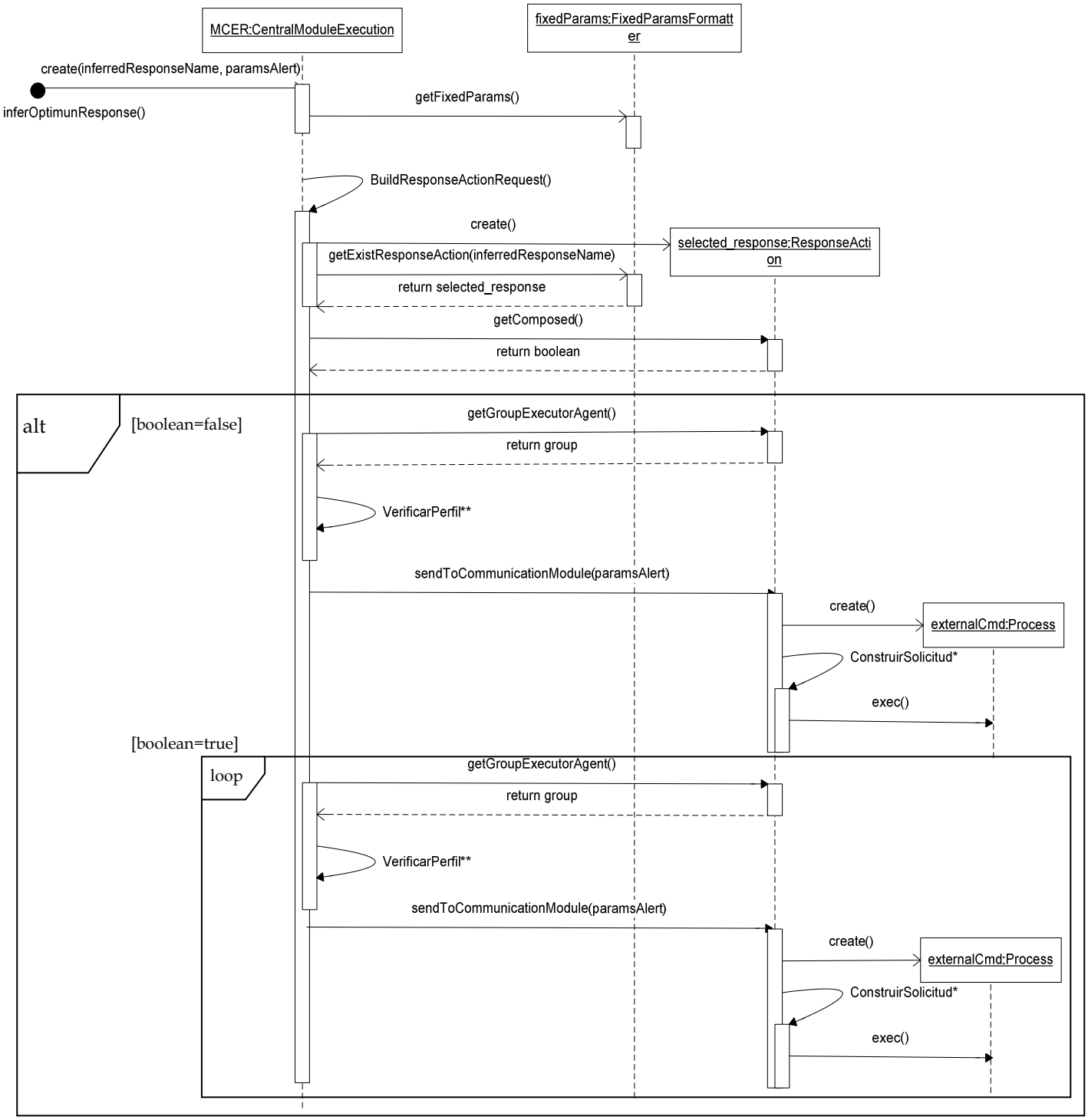



**Figura 30. Secuencia de operación del módulo central de ejecución.**

## 6.5 Módulo de comunicación y agente de ejecución

La funcionalidad requerida por el módulo de comunicación y el agente de ejecución, pueden ser provistas en su mayoría por la herramienta open source SnortSam y por consiguiente como parte del proceso de diseño, realizado en el capítulo 5, se decidió reutilizar dichos componentes. Por esta razón la implementación de ambos componentes se aborda en una sola sección; el propósito es mencionar únicamente aquellos cambios relevantes que se realizaron con el fin de adaptarlo al AIRS. El código fuente de SnortSam puede ser descargado de http://snortsam.net/download.html, se trabajó a partir de la versión 2.7 de esta herramienta.

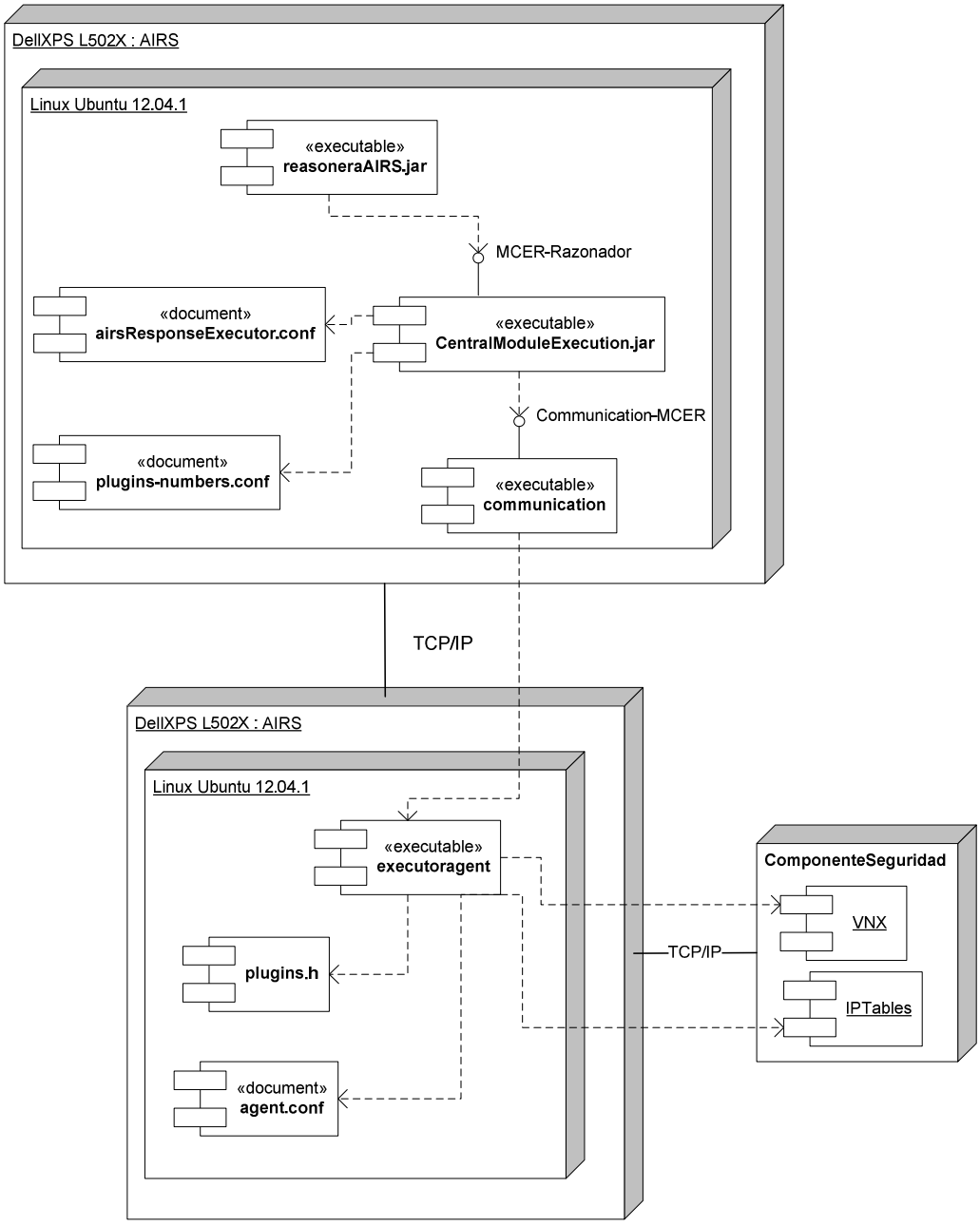



Figura 31. Diagrama de despliegue del ejecutor de respuestas.

### 6.5.1 Definición de la interfaz MCER-Communication

En la figura 31 se muestra el diagrama de despliegue completo del ejecutor de respuestas; como se observa, entre el módulo de comunicación y el MCER debe existir una interfaz de comunicación. No obstante, SnortSam está conectado a Snort como un plugin de salida. Consecuentemente, la primera tarea llevada a cabo en la implementación de este módulo fue la modificación y adaptación del código de tal forma que sea posible acoplarlo al MCER.

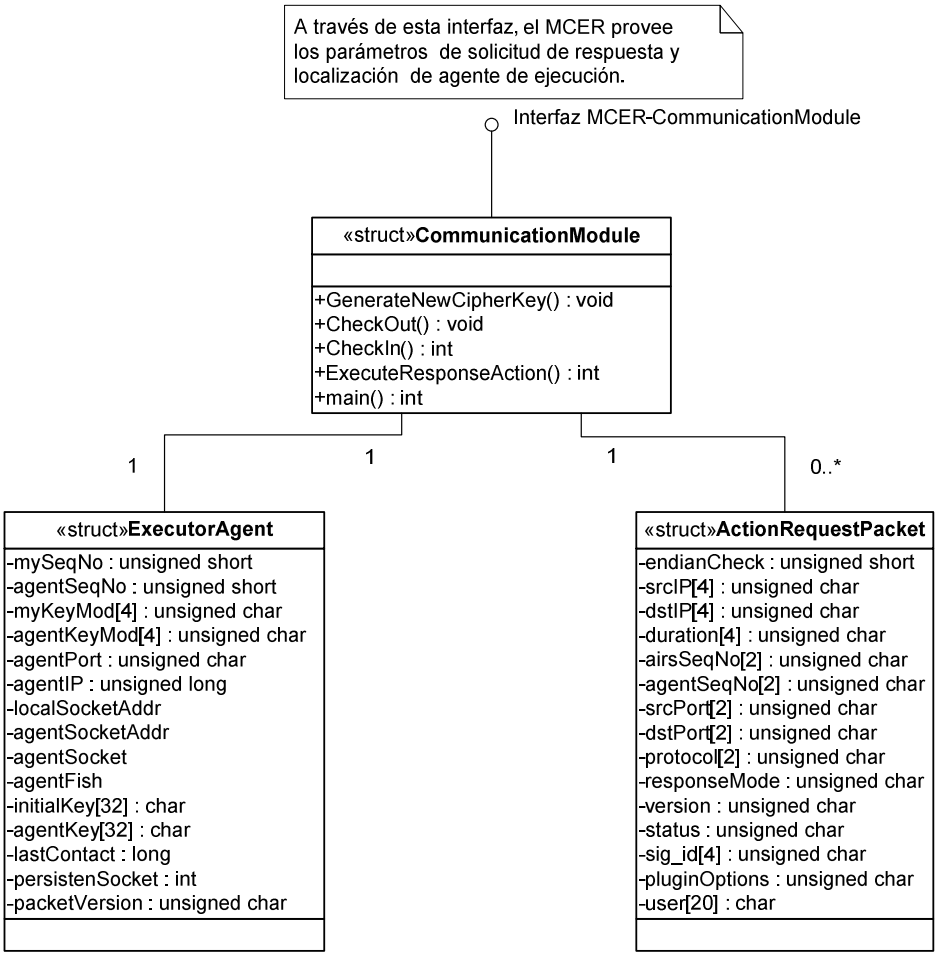


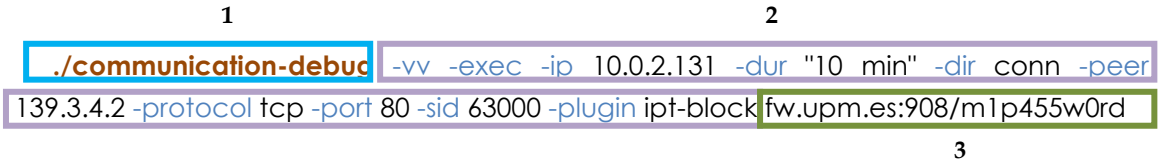

**Figura 32. Estructuras y funciones definidas en el módulo de comunicaciones.**

El *MCER* llama al *módulo de comunicación* como un comando externo, pudiéndose obtener también la salida estándar y la salida de error, producto de dicha ejecución.

```
cmd = "./communication-debug -vv -e " + cmd;
externalCmd = Runtime.getRuntime().exec(cmd);
```

Se ha adaptado SnortSam de tal forma que los parámetros necesarios para emitir la solicitud y la información de los agentes de ejecución sobre los que se ejecuta la respuesta sean provistos al módulo de comunicación como parámetros de entrada al método *main*.

Cómo se mencionó en la sección 6.4.2, a través de la respuesta seleccionada (ver Figura 30) se llama a la función *SendToCommunicationModule(paramsAlert)*. Quién hace uso del marco común de comunicación para emitir una solicitud de respuesta. En las sentencias que se muestran a continuación el MCER solicita al módulo de comunicación (1) la emisión de una solicitud de respuesta, al *agente de ejecución* que está sobre el firewall con nombre de host *fw.upm.es*, puerto *908 (3)* y password *m1p455w0rd*, en donde además incluye los parámetros necesarios para la ejecución (2).




En la Tabla 25 se muestran las directivas que han sido definidas, para el paso de parámetros hacia el módulo de comunicación.

| Directiva | Descripción | Observación |
|---|---|---|
| -e[xecute] | Solicitud de una acción de respuesta hacia al agente especificado. | |
| -un[do] | Solicitud para "deshacer" una acción de respuesta. | |
| -i[p] | Dirección IP del atacante. | |
| -pl[ugin] | Nombre del plugin que ejecuta la respuesta. | Debe coincidir con el definido en el registro del plugin. |
| -w[ho] | Sobre quién se ejecuta la respuesta. | **src**: sobre el atacante<br>**dst**: sobre la víctima |
| -du[ration] | Intervalo de tiempo que tiene efecto la acción de respuesta | |
| -dir[ection] | Dirección en la que se filtra el tráfico (para respuestas basadas en red) | **in**: Solo tráfico de entrada<br>**out**: Sólo tráfico de salida.<br>**inout:** Tráfico de entrada y salida<br>**this:** La conexión específica. |
| -sid | Identificador de intrusión. | |
| -pe[er] | Dirección IP de la víctima. | |
| -pr[oto] | Protocolo empleado para establecer la sesión. | |
| -po[rt] | Puerto de la víctima. | |
| -us[er] | Usuario utilizado para el ataque. | |
| -ad[ditional] | Parámetro adicional. | |
| host:puerto/contraseña | Información de los agentes de ejecución | Necesariamente deben ir al final de sentencia, y se puede especificar más de un agente. |

**Tabla 25. Directivas para el paso de solicitud desde el MCER hacia el módulo de comunicación.**

### 6.5.2 Agente de ejecución

Una de las ventajas de la reutilización de algunos componentes de SnortSam es que esta herramienta provee una variedad de capacidades que van más allá de la ejecución de un bloqueo sobre un firewall y que también son utilizadas por el módulo ejecutor de respuestas:

- Soporta la definición de listas blancas de direcciones IP que nunca serán bloqueadas.
- Se pueden definir tiempos de duración máximos o mínimos que tendrá efecto una acción de respuesta emitido por el MCER.
- Se puede definir una lista de SIDS asociados al MCER ante los cuales se ejecuta o no una acción de respuesta.
- Tiene la capacidad de ejecutar un rollback de acciones de respuesta cuando se ha identificado un ataque de Denegación de Servicio.
- Previene la emisión repetitiva de acciones de respuesta por medio de la definición de una ventana de tiempo entre peticiones. Esto mejora el rendimiento del ejecutor.
- Permite la expiración de una acción de respuesta luego de que ha transcurrido un intervalo de tiempo determinado.
- Permite el procesamiento multithreading para un procesamiento rápido y simultáneo de acciones de respuesta sobre múltiples dispositivos.

Cuando el *agente de ejecución* inicia, éste abre y parsea el archivo de configuración *agent.conf*. Éste archivo de configuración, propio de SnortSam, posee una gama de opciones para soporte de las funcionalidades antes mencionadas. En la Figura 33 se muestra un fragmento del archivo de configuración, en donde se define: la contraseña por defecto para cifrar y descifrar los mensajes, el puerto sobre el cual escucha las peticiones provenientes desde el MCER, las direcciones IP y contraseñas de los MCER a los cuales se les permite el envío de una solicitud, la definición de una lista blanca sobre los cuales no se ejecuta una acción de respuesta y los parámetros requeridos por un plugin en particular.

```
# agent.conf
#
# ExecutorAgent  options:
# -----------------------
# defaultkey <key>
defaultkey P3dfkh2443*33
# port <port>
port 34000
# accept <host>/<mask>,<key>
accept 138.100.211.28, mypassword
# dontexecute <host>/<mask>
dontundoexecute 192.168.10.0/24
#
# ResponseAction specific options:
# --------------------------
# hnvnx <pathFile><user>
hnvnx /usr/share/vnx/examples/tutorial_ubuntu.xml root
# ipt-block <adapter> <logoption>
ipt-block att-e0 138.100.211.28:25 danny.guamanl@gmail.com ids@upm.es
# email <smtpserver>:<port> <recipient> <sender>
email localhost:25 danny.guamanl@gmail.com  admin@alumnos.upm.es
# ciscoacl <ip> <telnetpw> <enablepw> <acl_filename>|[tftp_ip]: <acl_interface> <acl_type> <acl_name>
ciscoacl 10.1.110.3 cisco class | 10.1.110.5 : FastEthernet1/0 in AIRS
```

Figura 33. Archivo de configuración *agent.conf* del agente de ejecución.

### *Gestor de plugins*

Cómo se ha mencionado en varios puntos de la presente memoria, una de las características más importantes de la arquitectura propuesta, es la de proveer un marco común de comunicación entre el MCER y los agentes de ejecución; consecuentemente, el gestor de plugins también juega un rol importante en la arquitectura, al permitir un despliegue simple de nuevas acciones de respuesta de una forma "conectable". Para que aquello sea posible, se define una interfaz de registro a través del cual se debe proveer un conjunto de parámetros necesarios para la ejecución de una respuesta sobre un componente de seguridad determinado.

El fichero de cabecera *plugin.h* es el que permite el registro de un nuevo plugin a través de la estructura PLUGINREGISTRY:

```
typedef struct _plugins
{   int (*PluginInit)(DATALIST *);
    void (*PluginConfigParse)(char *,
          char *,unsigned long,
          DATALIST *);
    void (*PluginResponseAction)(RESPONSEACTIONINFO *,
                void *,unsigned long,unsigned short, char *);

    void (*PluginExit)(DATALIST *);
    void (*PluginKeepAlive)(DATALIST *);
    int PluginNeedsExpiration;
    int PluginDoesReblockOnSignal;
    int PluginThreading;
    char PluginHandle[40];
    unsigned short PluginNumber;
    char PluginAuthor[100];
    char PluginVersion[30];
} PLUGINREGISTRY;
```

En la Tabla 26 se hace una descripción de cada uno de los parámetros que han de proveerse para el registro de un nuevo plugin en el *agente de ejecución:*

| Nombre | Parámetros | Descripción |
|---|---|---|
| ***PluginInit** | **DATALIST *:** Puntero a la estructura que contendrá todos los componentes de seguridad asociados a este plugin. | Función que ejecuta una rutina de inicio requerida por el plugin. La función retorna TRUE si la rutina de inicio se ha ejecutado satisfactoriamente y FALSE en caso contrario. Si retorna FALSE, el agente de ejecución deshabilita el plugin.<br>Si no se requiere la rutina de inicio se establece a NULL. |
| ***PluginConfigParse** | **Char *:** Línea de configuración del fichero *agent.conf* a parsear.<br>**Char*:** Nombre fichero de configuración (*agent.conf por default*).<br>**Unsigned Long**: Número de línea que se está parseando.<br>**DATALIST ***: Puntero a la estructura que contendrá todos los componentes de seguridad asociados a este plugin. | Función que ejecuta una rutina para parsear la línea de configuración de este plugin, definido en el fichero *agent.conf*. Cada línea parseada corresponde a un nuevo componente de seguridad y se lo almacena en una estructura DATALIST.<br>Su implementación es obligatoria. |
| ***PluginResponseAction** | **RESPONSEACTIONINFO** *: Estructura que contiene los parámetros necesarios para ejecutar la acción de respuesta.<br>Void *: Puntero a la estructura **data** de DATALIST que contiene los parámetros específicos de un componente de seguridad asociado a este plugin.<br>Unsigned long: Para uso interno en caso de requerirlo.<br>Unsigned short: Para uso interno en caso de requerirlo.<br>Char: Nombre del plugin que lo llama. | Función que ejecuta la rutina de acción de respuesta sobre los componentes de seguridad definidos. Este fichero encapsula toda la lógica de control necesaria para ejecutar la respuesta sobre cada componente de seguridad; por consiguiente contiene los comandos y el medio de acceso propios de dicho componente.<br>Su implementación es obligatoria. |
| ***PluginExit** | **DATALIST** *: Puntero a la estructura que contiene todos los componentes de seguridad asociados a este plugin. | Función que ejecuta una rutina de salida, esta se ejecuta cuando SnortSam se cierra. Esto da al plugin la opción de limpiar o ejecutar algunas acciones a conveniencia.<br>Si no se requiere se establece a NULL. |
| ***PluginKeepAlive** | **DATALIST** *: Puntero a la estructura que contiene todos los componentes de seguridad asociados a este plugin. | Función que ejecuta una rutina de keep-alive contra componentes de seguridad que requieren una conexión persistente. Ejecutará esta función cada determinado tiempo. keep-alive.<br>Si no se requiere se establece a NULL. |
| **PluginNeedsExpiration** | | Indica si es necesario que el *agente de* |

| | |
|---|---|
| | *ejecución* controle la expiración de un bloqueo. TRUE indica que si es necesario, FALSE lo contrario.<br>Para proveer esta funcionalidad, el *agente de ejecución* crea un fichero de estado, de tal forma pueda deshacer una acción aún cuando se reinicie el agente. |
| **PluginThreading** | Determina si el plugin requiere de una ejecución multithread:<br>• 0: No se ejecuta en un nuevo hilo, sino que corre en línea con el proceso principal del agente de ejecución.<br>• 1: El plugin tiene capacidad multithreading, pero se contacta con sus dispositivos de forma secuencial: Es decir el plugin puede ejecutarse en paralelo con otros plugins, pero procesará un dispositivo a la vez.<br>• 2: El plugin tiene capacidad multithreading, se puede ejecutar más de una vez en hilos de ejecución diferentes, es decir en forma paralela con otros plugins al mismo tipo. |
| **PluginHandle[40]** | Identificador del plugin. Al definir los parámetros de este plugin en el archivo *agent.conf,* la línea de configuración debe iniciar con este identificador. |
| **PluginNumber** | Número identificador del plugin, es usado por el MCER, para especificar que plugin ejecutará la respuesta. |
| **PluginAuthor[100]** | Nombre del autor del Plugin. |
| **PluginVersion[30]** | Versión del plugin. |

**Tabla 26. Interfaz de registro de un nuevo plugin.**

El modo de operación general del gestor de plugins es el siguiente:

1. Cuando el *agente de ejecución* inicia y parsea el fichero *agent.conf*; tan pronto como alcanza la línea de configuración de un plugin, identificado por su *PluginHandle,* llama a cada una de las funciones de inicialización *PluginInit* de los plugins, registrados en el fichero *plugin.h*. A dicha función de inicialización se le pasa un puntero al primer elemento de la lista de estructuras DATALIST (la estructura de DATALIST se explica en el siguiente paso, dado que su uso es más común allí). Si la fase de inicialización es correcta entonces el plugin es habilitado,

de lo contrario es deshabilitado. Por ejemplo un plugin que despliega una *honeynet* usando la herramienta de virtualización VNX, verifica en primera instancia si se dispone de VNX sobre el host a ejecutar; en caso de no existir se devuelve *false* y el plugin es deshabilitado.

```
int HNVNXInit(DATALIST *datalistp){
    char *cmd = "vnx --version > /dev/null";
    if(!system(cmd))
        return 1;
    else
        return 0;
}
```

2. Luego de la inicialización, el gestor de plugins llama a la función de parsing *PluginConfigParsing*, definida también en el proceso de registro en el archivo *plugin.h*. A esta función se envía como parámetro, un puntero al primer elemento de la lista de estructuras DATALIST. Cada estructura de la lista DATALIST representa a un componente de seguridad y está constituida de 4 parámetros:

```
typedef struct _datalist
{   void *data;
    volatile unsigned long readpointer;
    volatile int busy;
    struct _datalist *next;
}   DATALIST;
```

   - **data*:** A este puntero se asigna una nueva estructura con los parámetros específicos de un componente de seguridad. Al ser de tipo void, brinda al desarrollador del plugin la libertad de definir dicha estructura en función de sus requerimientos.
   - **readpointer:** Es un puntero a la cola de solicitudes de acciones de respuesta, pendientes de ejecutar por el componente de seguridad.
   - **busy:** Es un flag para indicar que el componente de seguridad está ocupado ejecutando una acción de respuestas.
   - **next:** apunta a la próxima estructura DATALIST.

   Por ejemplo, para que un plugin pueda despeglar una honeynet usando VNX requiere básicamente dos parámetros: la ruta del fichero XML, que contiene el escenario virtual a desplegar; y el nombre del usuario. Entonces es necesario definir una estructura que almacene dichos parámetros:

```
typedef struct _vnxdata
{   char path[100];
    char user[20];
}   VNXDATA;
```

   Una vez que el agente de ejecución llame a la función *PluginParse*, la estructura VNXDATA será enlazada al puntero **data** de la estructura DATALIST, como se muestra a continuación:

```
void HNVNXParse(char *val,char *file,unsigned long line,DATALIST *plugindatalist)
{   VNXDATA *vnxp=NULL;
    char *p2, msg[STRBUFSIZE+2];

#ifdef FWSAMDEBUG
    printf("Debug: [hnvnx] Plugin Parsing...\n");
#endif
    if(*val)
    {   p2=val;
        while(*p2 && !myisspace(*p2)) p2++;
        if(*p2) *p2++ =0;
        vnxp=safemalloc(sizeof(VNXDATA),"HNVNXParse","vnxp");
        plugindatalist->data=vnxp;
        safecopy(vnxp->path,val);   /* save path of a XML file which configure a VNX Honeynet */
        if(*p2)
        {   while(*p2 && myisspace(*p2)) p2++;
            safecopy(vnxp->user,p2);        /* user */
        } else {
            safecopy(vnxp->user,HNVNX_USER_DEFAULT);    /* use default user */
        }
#ifdef FWSAMDEBUG
        printf("Debug: [hnvnx] Adding HNVNX Parameters: file \"%s\", user \"%s\"\n", vnxp->path, vnxp->user);
#endif
    }
    else
    {   snprintf(msg,sizeof(msg)-1,"Error: [%s: %lu] HNVNX defined without parameters!",file,line);
        logmessage(1,msg,"hnvnx",0);
    }
}
```

3. Después de que todos los parámetros han sido inicializados y parseados, el plugin está listo para recibir solicitudes de acción de respuesta.
4. Cuando el *agente de ejecución* recibe una solicitud de respuesta desde el MCER válido (*definido mediante opción accept en el archivo agent.conf*), selecciona el plugin a ejecutar y posteriormente llama a la función de ejecución *PluginResponseAction*. La estructura RESPONSEACTIONINFO con la información de la acción de respuesta, es enviada como parámetro y puede ser utilizada en caso de requerirlo.

```
typedef struct _actioninfo
{   unsigned long sig_id;
    unsigned long blockip;
    unsigned long peerip;
    time_t duration;
    time_t blocktime;
    unsigned short port;
    unsigned short proto;
    unsigned short mode;
    short executeFlag;
    unsigned short plugin;
    char user[20];
}   RESPONSEACTIONINFO;
```

### *Plugins*

Cómo se planteo en el capítulo 3, una de las limitaciones que presenta el módulo ejecutor de respuestas del AIRS basado en ontologías, es que posee un catálogo muy reducido de acciones de respuesta. En virtud de ello, aprovechando la arquitectura distribuida del ejecutor de respuestas propuesto en este trabajo de fin de máster; se han implementado algunas pruebas de concepto de acciones de respuesta basadas en host y basadas en red, que serán añadidas al catálogo. En la Tabla 27 se listan las acciones de respuesta implementadas.

| Ficheros fuente | Nombre del Plugin PluginHandle | Parámetros provistos en el RESPONSEACTIONINFO | Parámetros definidos en el fichero *agent.conf* | Descripción. |
|---|---|---|---|---|
| ssp_ciscoacl.c<br>ssp_ciscoacl.h | Ciscoacl | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo | - Dirección IP Router<br>- Contraseña telnet<br>- Contraseña enable<br>- Archivo inicial de ACLs<br>- Nombre ACL (opcional)<br>- Interfaz ACL<br>- Tipo de ACL:in/out<br>- IP servidor TFTP (opcional) | Este plugin tiene el objetivo añadir una lista de control de acceso a un router Cisco, que será empleado en el entorno de prueba. El mismo plugin puede ser empleado para añadir una ACL que filtre el tráfico de entrada, salida, entrada y salida, o de una conexión específica, asociadas a la dirección IP del atacante. Este plugin puede operar de dos formas: 1) Enviando comandos individuales hacia el router a través de una conexión remota, y;2) Enviando un archivo running-config modificado desde un servidor TFTP. |
| ssp_ipt-block.c<br>ssp_ipt-block.h | ipt-block | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo | -Interfaz Ethernet | Este plugin tiene por objetivo insertar una regla a la tabla FILTER de un firewall IPTables. . El mismo plugin puede ser empleado para añadir una regla que filtre el tráfico de entrada, salida, entrada y salida o de una conexión específica, asociadas a la dirección IP del atacante. |
| ssp_vnxhn.c<br>ssp_vnxhn.h | Vnxhn | | -Ruta del fichero VNX.<br>-Nombre de usuario. | Este plugin despliega un escenario de red virtual usando la herramienta de virtualización Virtual Network over LinuX (VNX). |
| ssp_ipt-redirect.c<br>ssp_ipt-redirect.h | ipt-redirect | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo | -Dirección IP hacia donde se redirige el tráfico.<br>-Puerto hacia donde se redirige el tráfico. | Este plugin, redirige el tráfico de entrada, salida, entrada y salida, o de una conexión específica hacia un host definido como parámetro. Es utilizado junto con el plugin vnxhn. |
| ssp_cisconullroute.c<br>ssp_cisconullroute.h | cisconull-route | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo | -Dirección IP Router<br>-Contraseña telnet<br>-Contraseña enable | Este plugin agrega una ruta nula a un router Cisco. El mismo plugin puede ser empleado para añadir una ruta nula asociada al tráfico de entrada, salida, entrada y salida, o a una conexión específica asociada a la dirección IP del atacante. |
| ssp_updown-ftp.c<br>ssp_updown-ftp.h | upload-backup-ftp | | -IP servidor FTP<br>-Nombre de usuario<br>-Contraseña de usuario<br>-Fichero origen<br>-Directorio destino | Este plugin tiene por objetivo respaldar un archivo definido por el usuario hacia un servidor FTP. Si existe un archivo con el mismo nombre guarda una copia como bakup. |
| ssp_updown-ftp.c<br>ssp_updown-ftp.h | download-backup-ftp | | -IP servidor FTP<br>-Nombre de usuario<br>-Contraseña de usuario<br>-Fichero origen<br>-Directorio destino | Este plugin tiene por objetivo restaurar un archivo desde un servidor FTP definido por el usuario. Si existe un archivo con el mismo nombre guarda una copia como bakup. |

| Ficheros fuente | Nombre del Plugin PluginHandle | Parámetros provistos en el RESPONSEACTIONINFO | Parámetros definidos en el fichero *agent.conf* | Descripción. |
|---|---|---|---|---|
| ssp_backup-mysql.c<br>ssp_backup-mysql.h | backup-mysql<br>restore-mysql | | -Servidor MySQL<br>-Usuario BDD<br>-Contraseña BDD<br>-Base de Datos.<br>-Nombre Fichero Backup | Este plugin tiene por objetivo respaldar una base de datos MySQL, empleada en el entorno de prueba. |
| ssp_disable-user.c<br>ssp_disable-user.h | disable-user | -Nombre de usuario | | Este plugin tiene por objetivo deshabilitar un usuario sobre un host Linux con el que se está perpetrando un ataque. |
| ssp_close-port.c<br>ssp_close-port.h | close-port | -Puerto<br>-Protocolo: TCP/UDP | | Este plugin ejecuta una acción que cierra un puerto sobre el host atacado. |
| ssp_host-deny.c<br>ssp_host-deny.h | host-deny | -IP Atacante | - Ruta del fichero host.deny | Este plugin agrega una regla en el fichero host.deny, para denegar el acceso hacia el host atacado. |
| ssp_host-null-route.c<br>ssp_host-null-route.h | host-null-route | -IP Atacante | | Este plugin ejecuta una acción que agrega una ruta nula en el host atacado. |
| ssp_close-connection.c<br>ssp_close-connection.h | close-connections | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo | | Este plugin cierra todas las conexiones establecidas por un host, que coincidan con la dirección IP del atacante, de la víctima, del atacante y la víctima o de la conexión específica. |
| ssp_whitelist.c<br>ssp_whitelis.h | Whitelist | - IP Atacante | - Ruta del fichero de configuración del agente de ejecución (agent.conf). | Este plugin agrega la dirección IP de un host a la lista blanca del agente de ejecución, sobre la que no se ejecutarán acciones de respuesta. |
| ssp_email.c<br>ssp_email.h | Email | - Dirección IP atacante<br>- Dirección IP víctima<br>- Puerto de la víctima.<br>- Protocolo<br>… | - IP del servidor SMTP.<br>- Puerto del servidor SMTP.<br>- Dirección de correo origen.<br>- Dirección de correo destino. | Este plugin tiene por objetivo informar al administrador de la red que se ha producido una intrusión, para ello envía un correo electrónico con toda la información obtenida de dicha intrusión. |

**Tabla 27. Pruebas de concepto de acciones de respuesta implementadas como plugins del agente de ejecución.**

Es importante mencionar, que las acciones de respuesta a implementar dependerán en gran medida de los recursos que se requiera proteger y de los componentes de seguridad que disponga una organización en particular. He ahí la gran ventaja de proveer un marco común de comunicación y una arquitectura basada en plugins que permitan el rápido despliegue de nuevas acciones de respuesta que interactúen con otros componentes de seguridad. Además de aprovechar las funciones que provee SnortSam que van más allá de la simple ejecución autónoma de acciones de respuesta como se mencionó en la sección 6.5.3.

El siguiente capítulo aborda las pruebas de validación de la arquitectura propuesta, a través de la ejecución de las acciones de respuesta sobre un entorno de prueba virtualizado.

# 7 Resultados y validación de la arquitectura propuesta

Las pruebas expuestas en este capítulo hacen énfasis en mostrar el flujo de operación de los componentes que conforman el ejecutor de respuestas propuesto una vez que ha sido acoplado al AIRS basado en ontologías. El objetivo es resaltar las ventajas de contar con un sistema ejecutor de respuesta a intrusiones distribuido, seguro y escalable, como parte del AIRS basado en ontologías. En virtud de ello se presentan tres posibles escenarios; en cada uno de estos escenarios se describen los aspectos más relevantes del flujo de operación. Además para corroborar los resultados se muestran las salidas generadas por cada módulo, aprovechando que cada uno de ellos incluye sentencias de depuración y registros de logs que servirán para mostrar el funcionamiento del ejecutor de respuestas.

## 7.1 Entorno de pruebas

La validación de la arquitectura del ejecutor de respuestas se llevó a cabo en una red ad hoc que se muestra en la Figura 34. En lo posible se ha intentado emular una topología de red que comúnmente es desplegada en las organizaciones; por consiguiente, se han definido varias subredes a la que se conectan hosts con diferentes sistemas operativos. La red está constituida de los siguientes componentes de red:

- Cuatro subredes que están conectadas a un router Cisco:
  - Una subred asignada a la Zona Desmilitarizada (DMZ) que está constituida por varios servidores externos: un servidor web y un conjunto de servidores que se encuentran dentro de un mismo host y que para las pruebas contiene Metasploitable.dcd
  - Una subred en la que se encuentran los servidores más importantes para la organización (INTNet3), en este caso cuenta con un servidor de BDD.
  - Dos subredes en la que están conectados ordenadores con S.O. Linux y Windows y que corresponden a los empleados de la organización (INTNet1 e INTNet2).
- Existen 5 IDSs desplegados en la red de prueba, el propósito de estos sistemas de detección de intrusiones es detectar y enviar un alerta de intrusión hacia el AIRS basado en ontologías
  - Tres de ellos son NIDS Snort (IDS-1, IDS-2 e IDS-3), y;
  - Dos son HIDS OSSEC (IDS-4 e IDS-5).
- Existe un firewall perimetral (IPTables) a través del cual atraviesa el tráfico de entrada y salida que se intercambia entre ordenadores de la Intranet e Internet.

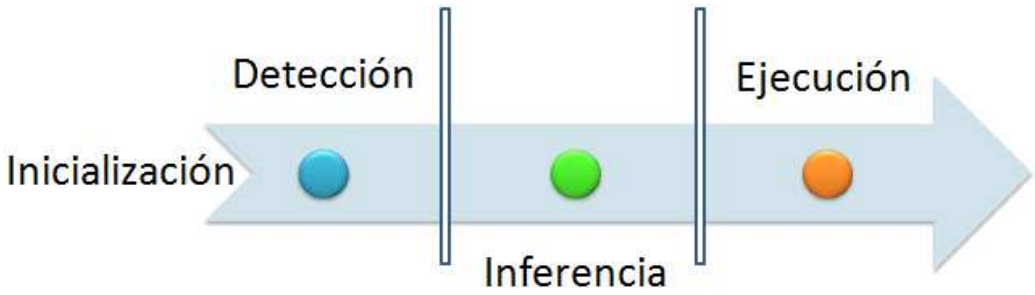
La topología de la red ha sido creada e implementada utilizando la herramienta de virtualización VNX (Virtual Network over LinuX)[21].

En la Tabla 28, se muestran detalles de cada uno de los componentes que se han empleado para implementar el entorno de pruebas.

| Componente | Características |
|---|---|
| RFW y RINT | Router Cisco C3640 Version 12.3(26) |
| ATT (Atacante) | GNU/Linux Ubuntu 10.04.3 LTS con BackTrack 5[22] |
| INT1-1, INT1-3, INT2-1 e INT2-5 | GNU/Linux Ubuntu 11.04 |
| INT1-2, INT1-4, INT2-2, INT2-3, INT2-4 e INT2-6 | Windows XP Service Pack 3 |
| INT3-1 e INT3-2 | GNU/Linux Ubuntu 11.04 |
| DMZ-1 | GNU/Linux Ubuntu 8.0.4 con Metasploitable[23] |
| DMZ-2 | GNU/Linux Ubuntu 11.04 |
| IDS-1, IDS-2 e IDS-3 | GNU/Linux Ubuntu 10.04.2 TTS<br>NIDS Snort versión 2.9.2.3[24] |
| IDS-4 e IDS-5 | GNU/Linux Ubuntu 11.04<br>HIDS OSSEC[25] versión 2.7 |
| AIRS basado en ontologías. | GNU/Linux Ubuntu 12.04.1 LTS<br>Ordenador: Dell XPS L502X, 8GB RAM, Procesador 2.6GHz. |

**Tabla 28. Detalles de los componentes empleados en el entorno de red.**

Con el propósito de mostrar los resultados obtenidos y la interacción entre los módulos propuestos e implementados, se divide el proceso en 4 actividades: inicialización, detección, inferencia y ejecución.




[21] http://web.dit.upm.es/vnxwiki/index.php/Main_Page
[22] http://www.backtrack-linux.org/
[23] http://sourceforge.net/projects/metasploitable/
[24] http://www.snort.org/snort-downloads/
[25] http://www.ossec.net/?page_id=19

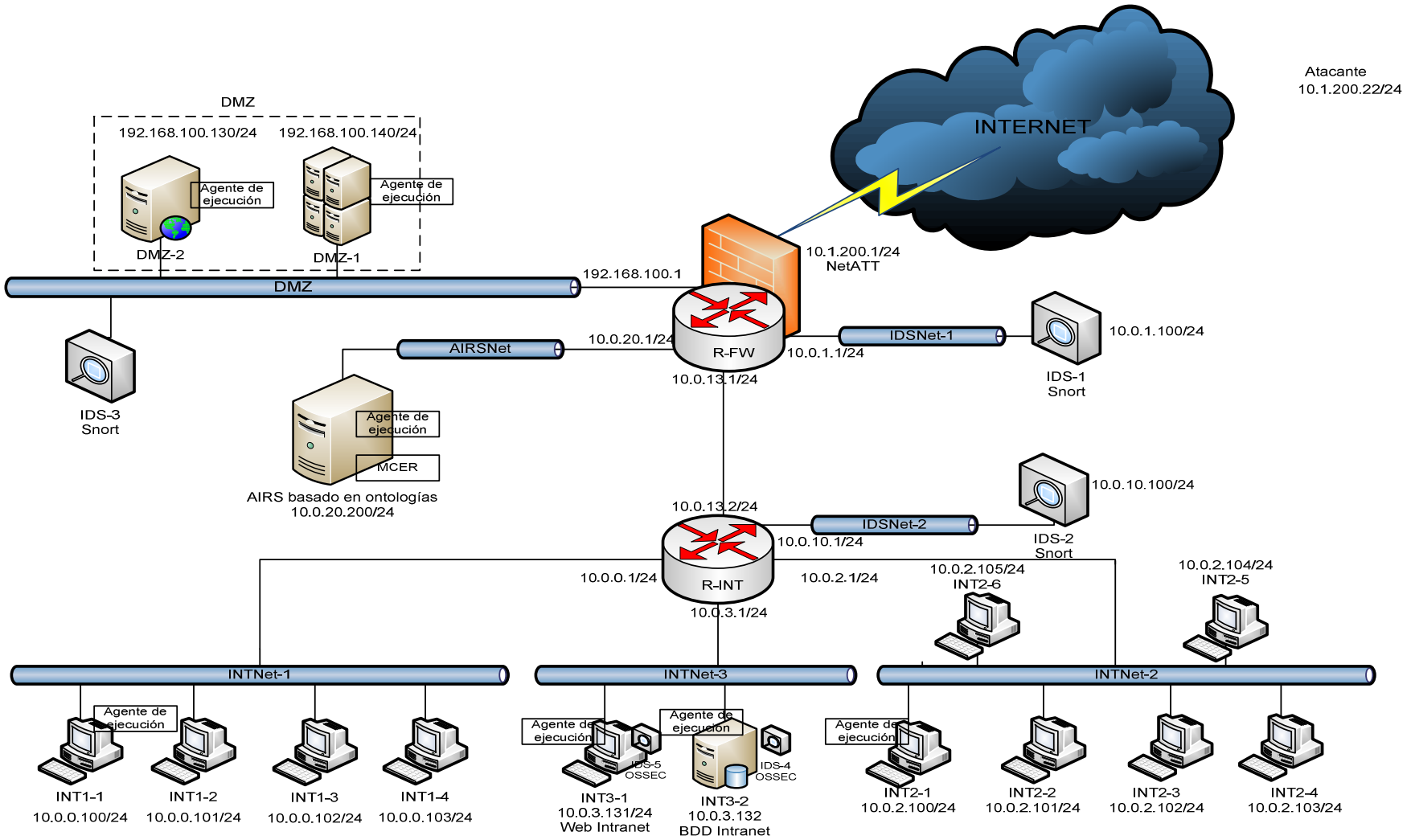



**Figura 34. Entorno de red de prueba.**

En la fase de inicialización, común para todos los escenarios, todos los componentes necesarios deben ser puestos en marcha. Dichos componentes los constituyen: los IDSs, el razonador AIRS, el MCER y los agentes de ejecución.

En la Tabla 29 se muestra la salida, producto de la inicialización del NIDS Snort y del HIDS OSSEC. En esta fase cada IDS lee sus archivos de configuración y queda listo para monitorizar las fuentes de datos respectivas y emitir las alertas de intrusiones en formato syslog hacia el puerto 512 del razonador AIRS.

| **Inicialización de OSSEC sobre el host 10.0.3.13 (INT3-2)** |
|---|
| root@INT3-S2:/var/ossec/bin# ./ossec-syscheckd -f -d<br>[…Líneas omitidas…]<br>2013/02/26 16:40:01 14:39:36 ossec-syscheckd: INFO: Monitoring directory: '/var/www'.<br>2013/02/26 16:40:01 14:40:38 ossec-syscheckd: INFO: Starting syscheck scan (forwarding database).<br>[…Líneas omitidas…]<br>2013/02/26 16:40:01 ossec-csyslogd: INFO: Started (pid: 15970).<br>2013/02/26 16:40:01 ossec-csyslogd: INFO: Forwarding alerts via syslog to: '10.0.1.1:512'. |
| **Inicialización de SNORT sobre el host 10.0.1.100 (IDS-1)** |
| root@vnx:/usr/local/snort/bin/snort –A console -ieth1 –c /usr/local/snort/etc/snort.conf -s<br>[...Líneas omitidas…]<br>Commencing packet processing (pid=1411) |

**Tabla 29. Salida-inicialización de IDSs.**

El detalle de la salida del *razonador AIRS* sale del alcance de la presente memoria. Por consiguiente, será considerada como una caja negra y se mostrará únicamente los resultados en sus interfaces de entrada y salida:

- La interfaz de entrada a través de la entrega de la alerta en formato syslog al *módulo receptor de alertas del AIRS.*
- La interfaz de salida MCER-Razonador a través de la cual luego de inferir la respuesta óptima, llama al MCER proporcionándole el nombre de la respuesta inferida y los parámetros respectivos.

El receptor de alertas escucha a través del puerto 512 de la interfaz con IP 10.0.1.1

| **Inicialización del razonador AIRS sobre el host 10.0.20.200 (AIRS)** |
|---|
| AlertReceptor is listening alerts...<br>_____<br>Start- method. SyslogAdapter<br>SyslogAdapter started.<br>Waiting syslog intrusion alerts on 10.0.1.1:512... |

**Tabla 30. Salida-inicialización del razonador AIRS.**

Junto con el razonador AIRS inicia el MCER, recordemos que en esta fase el MCER parsea dos ficheros de configuración: *plugin-numbers.conf y airsResponseExecutor.conf.* En la Tabla 31 se muestra la salida de la inicialización del MCER; el detalle del significado de cada parámetro fue abordado en los capítulos 5 y 6.

| **Inicialización del MCER sobre el host 10.0.20.200.** |
|---|
| **Debug:[MCER] Starting Central Module Execution...** |
| **Debug:[MCER] Parsing plugin-numbers.conf config file...** |
| Debug:[MCER] Adding a new plugin handler: ipt-block |
| Debug:[MCER] Adding a new plugin handler: hnvnx |
| Debug:[MCER] Adding a new plugin handler: email |
| Debug:[MCER] Adding a new plugin handler: email-execution-only |
| Debug:[MCER] Adding a new plugin handler: ciscoacl |
| Debug:[MCER] Adding a new plugin handler: ipt-redirect |
| Debug:[MCER] Adding a new plugin handler: disable-user |
| Debug:[MCER] Adding a new plugin handler: backup-mysql |
| Debug:[MCER] Adding a new plugin handler: restore-mysql |
| [...Líneas omitidas...] |
| Debug:[MCER] Adding a new plugin handler: upload-backup-ftp |
| Debug:[MCER] Adding a new plugin handler: download-backup-ftp |
| **Debug:[MCER] Parsing airsResponseExecutor.conf config file...** |
| Debug:[MCER] Parsing a new executor agent:R-FW/10.0.20.1, Port:34000, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new executor agent:INT3-1/10.0.3.131, Port:898, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new executor agent:INT3-2/10.0.3.132, Port:898, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new executor agent:HN-Server/127.0.0.1, Port:34001, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new executor agent:INT2-1/10.0.2.100, Port:898, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new executor agent:INT2-2/10.0.2.101, Port:898, Key:m1p455w0rd |
| [...Líneas omitidas...] |
| Debug:[MCER] Parsing a new executor agent:DMZ-2/192.168.100.130, Port:34000, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new group of executor agents:PerimetralFirewallGroup |
| 1.R-FW/10.0.20.1, Port:34000, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new group of executor agents:LocalAgentGroup |
| 1.local-agent/127.0.0.1, Port:898, Key:m1p455w0rd |
| Debug:[MCER] Parsing a new group of executor agents:IntranetCriticalHostGroup |
| 1.INT3-1/10.0.3.131, Port:898, Key:m1p455w0rd |
| 2.INT3-2/10.0.3.132, Port:898, Key:m1p455w0rd |
| [...Líneas omitidas...] |
| Debug:[MCER] Parsing a new SIDS group:BackdoorGroup: 32456 12343 12343 45633 45677 |
| Debug:[MCER] Parsing a new SIDS group:DoSGroup: 636 678 564 |
| Debug:[MCER] Parsing a new response action:blockInAttack, Type:Simple, Group Executors Agents:PerimetralFirewallGroup, Plugin:ipt-block, Who:src, Action:execute, Duration:24hours, Mode-Block:in, SIDs group:ALL |
| [...Líneas omitidas...] |
| Debug:[MCER] Parsing a new response action:mailNotification, Type:Simple, Group Executors Agents:LocalAgentGroup, Plugin:email, Who:src, Action:execute, Duration:0, Mode-Block:in, SIDs group:ALL |
| Debug:[MCER] Parsing a new response action:addACL, Type:Simple, Group Executors Agents:LocalAgentGroup, Plugin:ciscoacl, Who:src, Action:execute, Duration:2min, Mode-Block:in, SIDs group:ALL |
| Debug:[MCER] Parsing a new response action:honeynetDeployment, Type:Simple, Group Executors Agents:HoneyNetServerGroup, Plugin:hnvnx, Who:src, Action:execute, Duration:2hours, Mode-Block:0, SIDs group:DoSGroup |
| Debug:[MCER] Parsing a new response action:restoreIndex, Type:Simple, Group Executors Agents:DMZServer, Plugin:download-backup-ftp, Who:src, Action:execute, Duration:0, Mode-Block:0, SIDs group:ALL |
| Debug:[MCER] Parsing a new response action:blockInINTNet2, Type:Simple, Group Executors Agents:INTNet-2, Plugin:ipt-block, Who:src, Action:execute, Duration:2hours, Mode-Block:in, SIDs group:ALL |
| Debug:[MCER] Parsing a new response action:relayAttack, Type:Composed, Responses: redirectTraffic honeynetDeployment mailNotification restoreIndex |
| Debug:[MCER] Parsing a new response action:isolateHost, Type:Composed, Responses: blockInINTNet2 addACL |

**Tabla 31. Salida-Inicialización MCER.**

Finalmente, en la Tabla 32 se muestra la salida de la inicialización de los agentes de ejecución; en esta fase, el agente ejecuta básicamente dos acciones: el parsing del fichero de configuración agent.conf, y través del gestor de plugin, abordado en la

sección 6.5.3, ejecuta las funciones de inicialización *PluginInit* y de parsing *PluginConfigParsing*.

```
Info: [executoragent] Parsing config file agent.conf...
Info: [executoragent] Linking plugin 'hnvnx'...
Info: [executoragent] Initializing plugin 'hnvnx'...
Debug: [hnvnx] Plugin Parsing...
Debug: [hnvnx] Adding HNVNX Parameters: file "/usr/share/vnx/examples/honeynet1.xml", user "root"
Info: [executoragent] Linking plugin 'ipt-block'...
Debug: [ipt-block] Plugin Parsing...
Debug: [ipt-block] Adding IPT-BLOCK: interface "att-e0"
Info: [executoragent] Linking plugin 'email'...
Debug: [email] Plugin Parsing...
Debug:[email] Adding email plugin parameters: SMTP Server: 127.0.0.1, Recipient: danny.guamanl@gmail.com, Sender: admin@upm.es
        [...líneas omitidas...]
Debug: [executoragent] Starting to keep track of blocks regardless of plugins used in file /var/db/executoragent.state.
Info: [executoragent] Checking for existing state file "/var/db/executoragent.state".
Info: [executoragent] Found. Reading state file.
Debug: [executoragent] Accepting connections from:
Debug: [executoragent] IP: 10.0.20.200, Mask: 255.255.255.255, Pass: m1p455w0rd
Debug: [executoragent] Dontexecute list:
Debug: [executoragent] IP: 192.168.100.130, Mask: 255.255.255.255 (execute-action)
Debug: [executoragent] IP: 192.168.100.140, Mask: 255.255.255.255 (execute-action)
Debug: [executoragent] OnlyEXECUTE list:
Starting to listen for AIRSExecutor alerts
```

Tabla 32. Salida-Inicialización Agente de Ejecución.

## 7.2 Escenario 1: Respuesta activa de protección a un taque externo

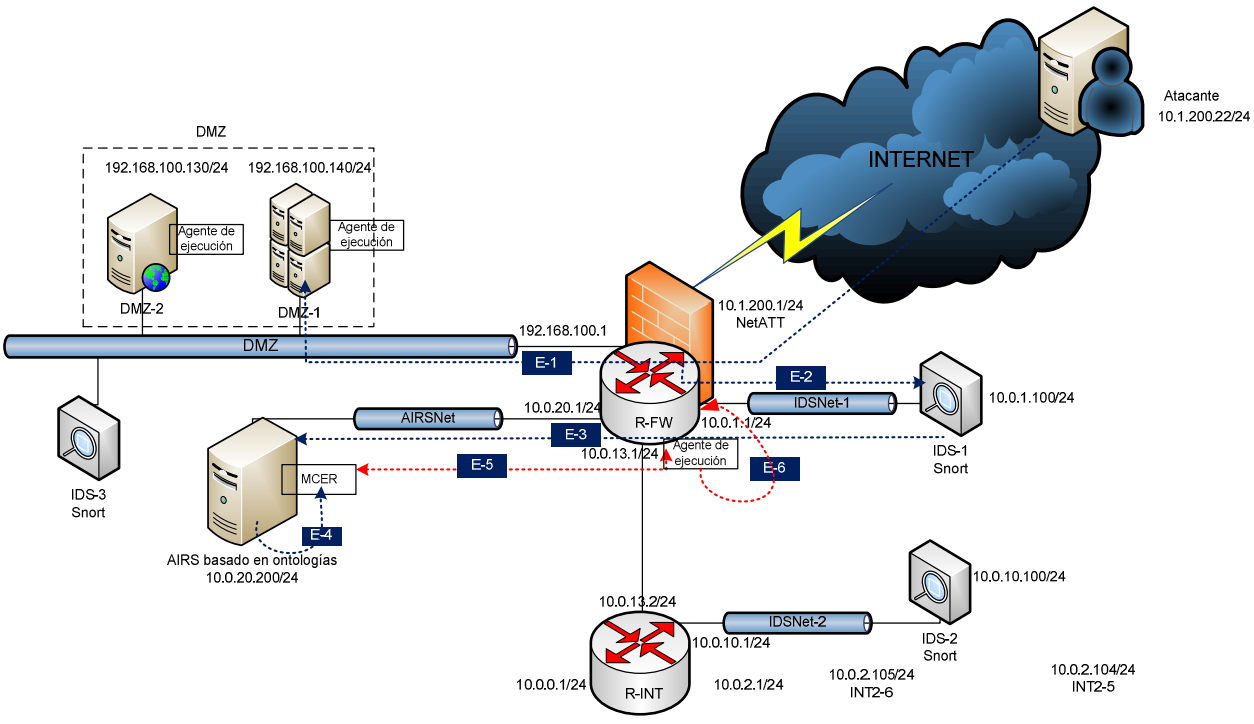



Figura 35. Respuesta activa de protección a un ataque desde el exterior de la red.

### 7.2.1 Detección

- **E-1**: El atacante (10.1.200.22), como "primera fase" de un ataque, intenta realizar el reconocimiento del objetivo, en este caso el servidor DMZ-1 (192.168.100.140). Para ello, ejecuta un escaneo de puertos utilizando la herramienta *Zenmap*[26], interfaz gráfica de nmap incluida en *BackTrack 5*, dirigida al servidor *DMZ-1*.
- **E-2:** Todo el tráfico que ingresa a través de la interfaz NetATT, como se explicó en la sección 6.2 es reescrito hacia el NIDS Snort IDS-1 por medio de la TAP daemonlogger.
- **E-3:** El NIDS Snort detecta la intrusión (ver Figura 36) y al mismo tiempo emite una alerta en formato syslog hacia el razonador AIRS.

```
03/05-18:37:56.191474  [**] [1:469:3] ICMP PING NMAP [**] [Classification: Attem
pted Information Leak] [Priority: 2] {ICMP} 10.1.200.22 -> 192.168.100.140
03/05-18:40:03.398093  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Atte
mpted Information Leak] [Priority: 2] {TCP} 10.1.200.22:57166 -> 192.168.100.140
:1
```

Figura 36. Detección de escaneo de puertos 10.1.200.22 -> 192.168.100.140.

El razonador AIRS, escucha a través de su módulo *de recepción de alertas* en el puerto 512 y recibe las alertas en formato syslog.

### 7.2.2 Inferencia

Posterior a la recepción de la alerta, el razonador AIRS infiere la respuesta óptima (ver Tabla 33). El proceso de inferencia fue revisado en detalle en la sección 2.4. En este caso el razonador infiere una respuesta activa de protección, cuya acción bloquea el tráfico de entrada del atacante en el firewall perimetral. Para la ejecución de la respuesta llama al módulo central de ejecución de respuestas (MCER), a quien provee el nombre de respuesta inferida, en este caso *blockInAttack*, y los parámetros requeridos: la dirección IP del atacante (10.1.200.22), la dirección IP del objetivo del ataque (192.168.100.140) y el tipo de protocolo (ICMP).

| |
|---|
| **Syslog alert received.** |
| <38>snort: [1:469:3] ICMP PING NMAP [Classification: Attempted Information Leak] [Priority: 2] {ICMP} 10.1.200.22 -> 192.168.100.140 |
| **Inferring optimal response to a type 'InformationGathering' intrusion...** |
| **Responses allowed**: blockInAttack, blockOutAttack, blockInOutAttack, addACL, denyConnectionHost, redirectTraffic, honeynetDeployment, relayAttack, mailNotification, |
| **Parameters received**--> HIDS:false, Attacker_IP:10.1.200.22, Victim_IP:192.168.100.140, Attacker_port:null, Victim_port:null, Protocol:ICMP, IntrusionType: InformationGathering, SID:1:469:3 |
| **Inferred response**: blockInAttack |

Tabla 33. Salida-Inferencia de respuesta de óptima para un ataque de tipo InformationGathering.

[26] http://nmap.org/book/zenmap.html Último acceso:20/02/2013

### 7.2.3 Ejecución

- **E-4:** El MCER entra en escena, recibe la solicitud de respuesta desde el razonador AIRS y recupera toda la información necesaria para ejecutar la respuesta *blockInAttack*. Recordemos que dicha información ha sido parseada en la fase de inicialización.

```
Debug:[MCER] Parsing a new group of executor agents:PerimetralFirewallGroup
        1.R-FW/10.0.20.1, Port:34000, Key:m1p455w0rd

Debug:[MCER] Parsing a new response action:blockInAttack, Type:Simple, Group
Executors Agents:PerimetralFirewallGroup, Plugin:ipt-block, Who:src, Action:execute,
Duration:24hours, Mode-Block:in, SIDs group:ALL
```

Dicha información señala que: 1) la respuesta contiene una sola acción, 2) Se ejecuta sobre el *agente de ejecución* ubicado en firewall perimetral (10.0.20.1), 3) El plugin que lo ejecuta es el *ipt-block,* 4) La acción se ejecuta sobre el atacante, 5) La acción tiene un efecto de 24 horas, 6) La acción opera sobre el tráfico de entrada, 7) Esta acción se ejecuta independiente del identificador de alerta.

- **E-5:** El MCER a través del módulo de comunicación se conecta hacia el agente de ejecución (10.0.20.1) y envía un paquete de solicitud de respuesta. Ver Tabla 34.

```
[5/3/2013 1:0:51] [3] [MCER] Info:[MCER] New request for execution of response action:
10.1.200.22->192.168.100.140:0 Protocol:ICMP Intrusion Signature ID (SID):0
Debug:[MCER] Selected response action:blockInAttack
[5/3/2013 1:0:51] [3] [MCER] Info:[Main] Trying to send 'blockInAttack' response action request
to communication module...
Info: [CheckIn] Connected to host 10.0.20.1.
Info: [ExecuteResponseAction] Connected to host 10.0.20.1. Executing action IP 10.1.200.22.
Info: [CheckOut] Disconnecting from host 10.0.20.1.
_____:::::::::::::::::::::::::_____-- esperando alertas nuevas
```

**Tabla 34. Salida-Envío de solicitudes a los agentes de ejecución a través del módulo de comunicación.**

Previo a la ejecución de la acción de respuesta (en la fase inicialización) el plugin *ipt-block* parsea los parámetros necesarios para su ejecución; en este caso se parsea la interfaz sobre la cual se agrega la regla de filtrado:

```
Info: [executoragent] Linking plugin 'ipt-block'...
Debug: [ipt-block] Plugin Parsing...
Debug: [ipt-block] Adding IPT-BLOCK: interface "NetATT"
```

El agente de ejecución recibe la petición proveniente desde el MCER y verifica que fue originado en el MCER autorizado. En caso de una autenticación exitosa, descifra el paquete y determina que plugin será utilizado para interactuar con el componente de seguridad (ver Tabla 35).

```
Starting to listen for AIRSExecutor alerts
Debug: [executoragent] Connection from: 10.20.0.200.
Debug: [executoragent] Adding sensor 10.20.0.200 to list.
        [...Líneas omitidas...]
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] Deploying Plugin: 2
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] Executing response action on host 10.1.200.22 inbound for 86400 seconds
(Sig_ID: 0).
Debug: [ipt-block] Plugin Blocking...
Debug: [ipt-block] command  /sbin/iptables -I FORWARD -i att-e0  -s 10.1.200.22 -j DROP
Debug: [ipt-block] command2 /sbin/iptables -I INPUT -i att-e0  -s 10.1.200.22 -j DROP
Debug: [executoragent] Connection from: 10.20.0.200.
Debug: [executoragent] Received Packet: CHECKOUT
```

Tabla 35. Salida-Ejecución de plugin ipt-block para bloqueo de tráfico de entrada

- **E-6:** Como se observa en la Tabla 35, una vez que se identifica el plugin (está contenido en el paquete de solicitud) se llama a la función *IPTBResponseAction,* que interactúa con el componente de seguridad, en este caso ejecuta comandos de IPTables para el bloqueo del tráfico de entrada del atacante (10.2.200.22) en la interfaz ***att-e0.***

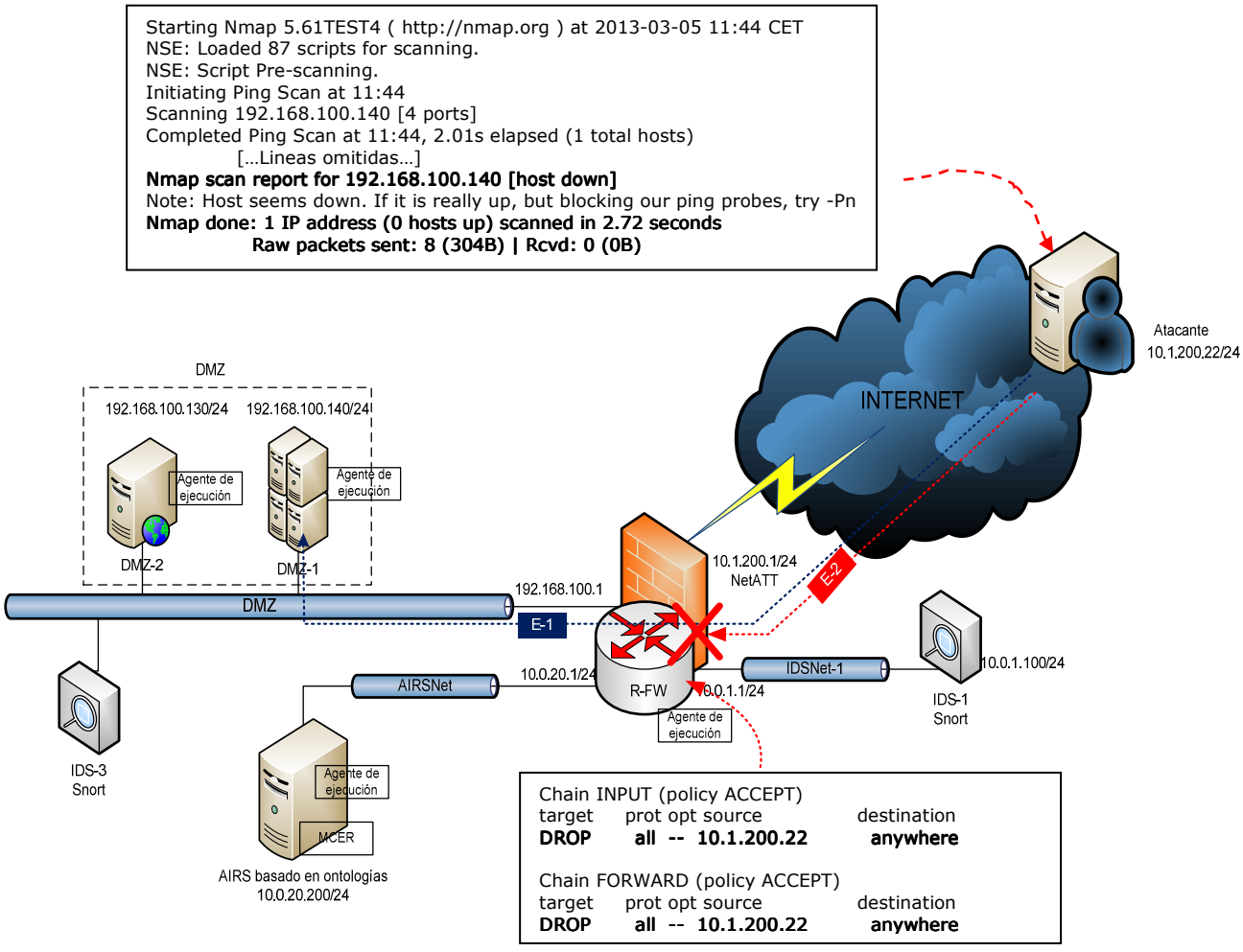



Figura 37. Resultado de la ejecución de la acción de respuesta.

En la Figura 37 se puede observar el resultado de la ejecución de la respuesta activa de protección, el firewall perimetral bloquea todo el tráfico proveniente del atacante (10.1.200.22). Para ello, agrega una regla de bloqueo a las cadenas INPUT y FORWARD de la tabla FILTER del firewall IPTables.

## 7.3 Escenario 2: Respuesta activa de protección a un ataque interno

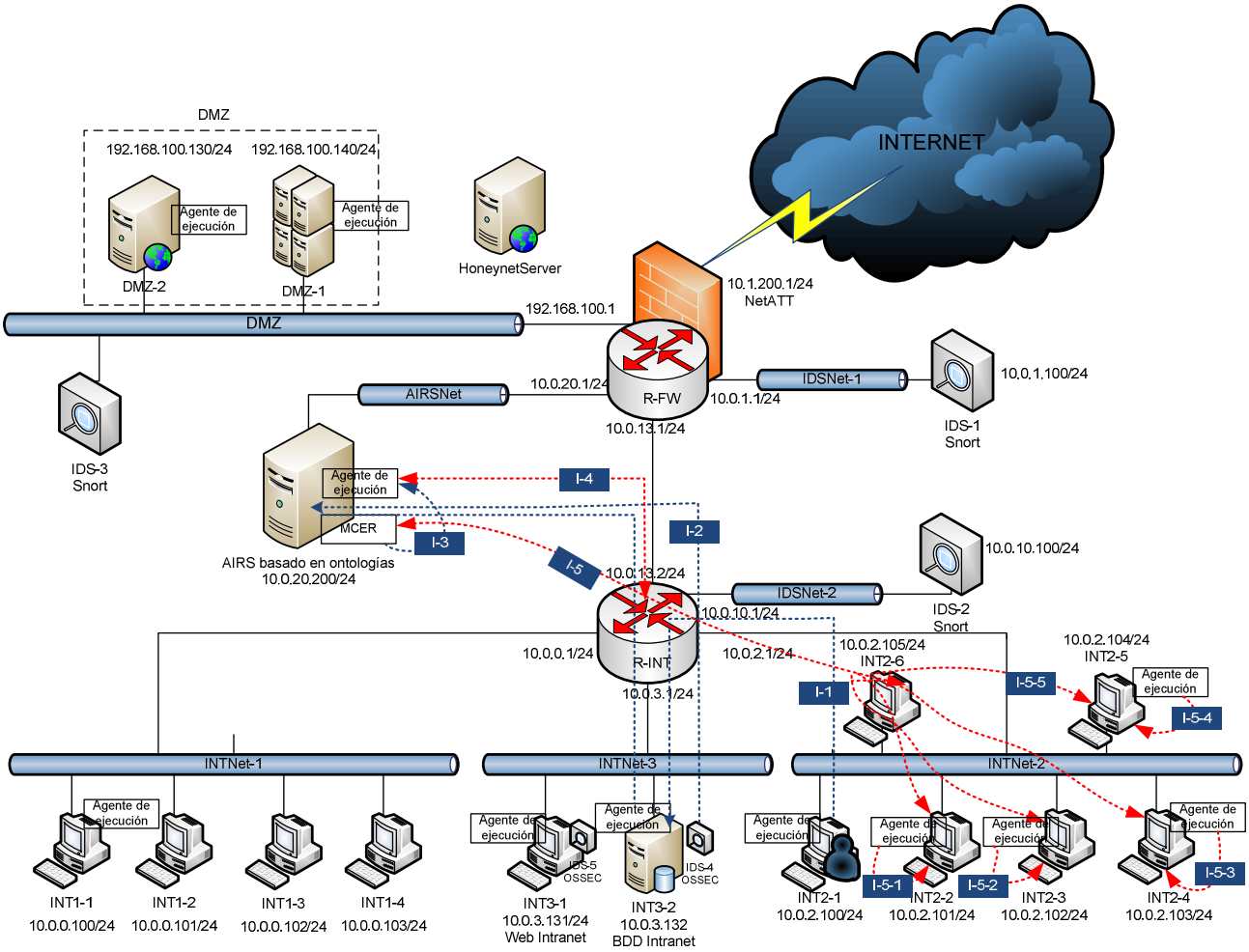



Figura 38. Respuesta activa de protección a un ataque desde el interior de la red..

### 7.3.1 Detección

- **I-1**: El atacante (10.0.2.100) ubicado en la red interna INTNet-2, intenta establecer una conexión ssh, por medio de un ataque de fuerza bruta, hacia el servidor de BDD de la intranet (10.0.3.132).
- **I-2:** Al ser el servidor de BDD un host crítico, se ha instalado el HIDS OSSEC. OSSEC, como se mencionó 6.2.2, realiza la verificación de integridad de ficheros, monitorización y análisis de logs del sistema, y detección de rootkits. En virtud de ello, al detectar el ataque emite una alerta de intrusión (ver Tabla 36) en formato syslog hacia el razonador AIRS.

```
root@INT3-S2:/var/ossec/bin# tail -n 5 ../logs/alerts/alerts.log
2013 Mar 07 17:52:09 INT3-S2->/var/log/auth.log
Rule: 5706 (level 6) -> 'SSH insecure connection attempt (scan).'
Src IP: 10.0.2.100
Mar  7 17:52:09 INT3-S2 sshd[1477]: Did not receive identification string from 10.0.2.100
```

Tabla 36. Detección de un ataque de fuerza bruta ssh 10.0.2.100 -> 10.0.3.132.

El razonador AIRS, escucha a través de su módulo *de recepción de alertas* en el puerto 512 y recibe las alertas en formato syslog.

### 7.3.2 Inferencia

Luego de la recepción de la alerta, el razonador AIRS infiere la respuesta óptima (ver Tabla 37). En este caso el razonador infiere una respuesta activa de protección, de nombre *isolateHost,* que tiene por objetivo *aislar al host atacante*. La respuesta inferida está compuesta por dos acciones simples: 1) la acción *addACL,* que agrega una regla a la lista de control de acceso del router interno (R-INT), y; 2) la acción *blockInINTNet,* que agrega una regla de filtrado al firewall personal de cada host que pertenece a la misma subred del atacante. Para la ejecución de la respuesta llama al módulo central de ejecución de respuestas (MCER), a quien provee el nombre de respuesta inferida (*isolateHost*), y los parámetros requeridos; en este caso únicamente la dirección IP del atacante (10.0.2.100).

**Syslog alert received.**
<132>Mar 7 17:52:09 INT3-S2 ossec: Alert Level: 6; Rule: 5706 - SSH insecure connection attempt (scan).; Location: INT3-S2->/var/log/auth.log; srcip: 10.0.2.100; Mar 7 17:52:09 INT3-S2 sshd[1477]: Did not receive identification string from 10.0.2.100
**Inferring optimal response to a type 'Guessing' intrusion...**
**Responses allowed**: addACL, isolateHost ,redirectTraffic,denyConnectionHost,mailNotification,
**Parameters received**--> HIDS:true, Attacker_IP:10.0.2.100, Victim_IP:INT3-S2, Attacker_port:null, Victim_port:null, Protocol:null, IntrusionType:Guessing, SID:5706
**Inferred response**: isolateHost

Tabla 37. Salida-Inferencia de respuesta de óptima para un ataque de tipo Guessing.

### 7.3.3 Ejecución

- **I-3:** El MCER recibe la solicitud de respuesta desde el razonador AIRS y recupera toda la información necesaria para ejecutar la respuesta *isolateHost*. La información asociada a ésta respuesta se muestra a continuación:

```
Debug:[MCER] Parsing a new group of executor agents:LocalAgentGroup
        1.local-agent/127.0.0.1, Port:898, Key:m1p455w0rd
Debug:[MCER] Parsing a new group of executor agents:INTNet-2
        1.INT2-2/10.0.2.101, Port:898, Key:m1p455w0rd
        2.INT2-3/10.0.2.102, Port:898, Key:m1p455w0rd
        3.INT2-4/10.0.2.103, Port:898, Key:m1p455w0rd
        4.INT2-5/10.0.2.104, Port:898, Key:m1p455w0rd
        5.INT2-6/10.0.2.105, Port:898, Key:m1p455w0rd

Debug:[MCER] Parsing a new response action:isolateHost, Type:Composed, Responses:
```

**addACL blockInINTNet2**

Debug:[MCER] Parsing a new response action:**addACL**, Type:**Simple**, Group Executors Agents:**LocalAgentGroup**, Plugin:**ciscoacl**, Who:**src**, Action:**execute**, Duration:**2min**, Mode-Block:**in**, SIDs group:**ALL**

Debug:[MCER] Parsing a new response action:**blockInINTNet2**, Type:**Simple**, Group Executors Agents:**INTNet-2**, Plugin:**ipt-block**, Who:**src**, Action:**execute**, Duration:**2min**, Mode-Block:**in,** SIDs group:**ALL**

Dicha información señala que la respuesta *isolateHost* está compuesta por dos acciones: *addACL y blockInINTNet2.*

- La acción *addACL*, se ejecuta sobre el *agente de ejecución* local. Dicho agente interactúa con un router Cisco por medio del plugin *cisoacl*, añadiendo una regla para el filtrado del tráfico del atacante.
- La acción *blockInINTNet2,* se ejecuta sobre los *agentes de ejecución* de los 5 hosts pertenecientes a la subred del atacante (INTNet-2). Cada agente interactúa con el el firewall IPTables por medio del plugin *ipt-block,* añadiendo una regla de filtrado del tráfico del atacante.

Debido a que la respuesta a ejecutar involucra a 6 agentes de ejecución, el MCER a través del módulo de comunicación debe establecer 6 conexiones y emitir las solicitudes de respuesta correspondientes. Ver Tabla 38.

```
[7/2/2013 7:30:16] [3] [MCER] Info:[MCER] New request for execution of response action: 10.0.2.100->INT3-S2:null Protocol:null Intrusion Signature ID (SID):0
Debug:[MCER] Selected response action:isolateHost
[7/2/2013 7:30:16] [3] [MCER] Info:[Main] Trying to send 'isolateHost' composite response action to communication module...
Info: [CheckIn] Connected to host 127.0.0.1.
Info: [ExecuteResponseAction] Connected to host 127.0.0.1. Executing action IP 10.0.2.100.
Info: [CheckOut] Disconnecting from host 127.0.0.1.
Info: [CheckIn] Connected to host 10.0.2.101.
Info: [ExecuteResponseAction] Connected to host 10.0.2.101. Executing action IP 10.0.2.100.
Info: [CheckOut] Disconnecting from host 10.0.2.101.
Info: [CheckIn] Connected to host 10.0.2.102.
Info: [ExecuteResponseAction] Connected to host 10.0.2.102. Executing action IP 10.0.2.100.
Info: [CheckOut] Disconnecting from host 10.0.2.102.
        [...Líneas omitidas...]
Info: [CheckIn] Connected to host 10.0.2.105.
Info: [ExecuteResponseAction] Connected to host 10.0.2.105. Executing action IP 10.0.2.100.
Info: [CheckOut] Disconnecting from host 10.0.2.105
_____:::::::::::::::::::::::::_____-- esperando alertas nuevas
```

**Tabla 38. Salida-Envío de solicitudes a los agentes de ejecución a través del módulo de comunicación.**

Previo a la ejecución de la acción de respuesta (en la fase inicialización) ambos plugins involucrados, parsean los parámetros necesarios para su ejecución. El plugin *ciscoacl* obtiene: la dirección IP del router, dirección IP del servidor TFTP, las contraseñas necesarias, el nombre de la ACL, el tipo de ACL y la interfaz en la que

se aplicará. Por su parte, el plugin *ipt-block* parsea la interfaz sobre la cual se agrega la regla de filtrado:

| **Sobre el agente de ejecución local:** |
|---|
| Info: [executoragent] Linking plugin 'ciscoacl'...<br>Info: [executoragent] Initializing plugin 'ciscoacl'...<br>Debug: [ciscoacl] Plugin Parsing...<br>Debug: [ciscoacl] TFTP Server defined: 10.0.13.1. Plugin will send a modified running-config file for each Response Action!<br>Debug: [ciscoacl]Adding CISCOACL plugin parameters. ACL Name:AIRS-ACL, Router: 10.0.2.1, Interface: Ethernet0/1, Type: in |
| **Sobre los agentes de ejecución INT2-2, INT2-3, INT2-4, INT2-5, INT2-6:** |
| Info: [executoragent] Linking plugin 'ipt-block'...<br>Debug: [ipt-block] Plugin Parsing...<br>Debug: [ipt-block] Adding IPT-BLOCK: interface "eth1" |

- **I-4:** Como se observa en la Tabla 39, el agente de ejecución local a través del plugin *ciscoacl* se comunica con el router R-INT (10.0.2.1) y realiza las siguientes tareas:
    - El momento de llamar la función de parsing, descarga el fichero runnig-config al servidor TFTP.
    - Sobre dicho fichero se busca si se ha definido antes una acl con el mismo nombre que el definido como parámetro (AIRS-ACL en este caso).
        - Si ya existe una ACL definida, entonces se agrega una nueva en el lugar adecuado de dicha lista.
        - Si no existe la ACL, entonces se crea una nueva ACL al fichero.
    - Se establece una conexión con el router y posteriormente se envían los datos para la autenticación.
    - Finalmente se sube el nuevo fichero running-config hacia el router desde el servidor FTP.

Starting to listen for AIRSExecutor alerts
Debug: [executoragent] Connection from: 127.0.0.1.
Debug: [executoragent] Adding sensor 127.0.0.1 to list.
Debug: [executoragent] Received Packet: CHECKIN
Debug: [executoragent] AIRSExecutor SeqNo: bd5f
Debug: [executoragent] ExecutorAgent SeqNo : 343e
Debug: [executoragent] Status : 1
Debug: [executoragent] Version : 15
Debug: [executoragent] Connection from: 127.0.0.1.
[…Líneas omitidas…]
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] **Deploying Plugin: 6**
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] **Executing response action on host 10.0.2.100 inbound for 7200 seconds (Sig_ID: 0).**
Executing Response Action: **Adding ACL Rule AIRS-ACL to 10.0.2.1**

```
Debug: [ciscoacl] ACL existence check:deny ip host 10.0.2.100 any
Debug: [ciscoacl] ACL Rule: deny ip host 10.0.2.100 any  isn't present!
Debug: [ciscoacl] Receiving: --Password: --
Debug: [ciscoacl] Sending:cisco
Debug: [ciscoacl] Receiving: -->--
Debug: [ciscoacl] Sending:enable
Debug: [ciscoacl] Receiving: --Password: --
Debug: [ciscoacl] Sending:class
Debug: [ciscoacl] Receiving: --#--
Debug: [ciscoacl] Sending:copy tftp: running-config
Debug: [ciscoacl] Receiving: --]? --
Debug: [ciscoacl] Sending:10.0.13.1
Debug: [ciscoacl] Receiving: --]? --
Debug: [ciscoacl] Sending:initial_acl.cfg_upload
Debug: [ciscoacl] Receiving: --]? --
Debug: [ciscoacl] Sending:running-config
Debug: [ciscoacl] Receiving: --#--
Debug: [ciscoacl] Response action executed successfully
```

Tabla 39. Salida-Ejecución del plugin ciscoacl para filtrado de tráfico de entrada.

- **I-5 (1-5):** Por cada uno de los hosts pertenecientes a la misma subred del atacante, el plugin *ipt-block* añade una regla de bloqueo a las cadenas INPUT y FORWARD de la tabla Filter de IPTables. En la Tabla 40 se muestra la salida, producto de la ejecución del plugin *ipt-block* sobre el host INT2-2 (10.0.2.101). El mismo resultado se obtiene para los hosts restantes.

```
Info: [executoragent] Executing response action on host 10.0.2.100 inbound for 7200 seconds (Sig_ID:
 0).
Debug: [ipt-block][b7849b70] Plugin Blocking...
Debug: [ipt-block][b7849b70] command  /sbin/iptables -I FORWARD -i eth1  -s 10.0.2.100 -j DROP
Debug: [ipt-block][b7849b70] command2 /sbin/iptables -I INPUT -i eth1  -s 10.0.2.100 -j DROP
Debug: [executoragent] Connection from: 10.0.13.1.
Tamaño del paquete recibido: 86
Tamaño a recibir: 86
Debug: [executoragent] Received Packet: RESPONSEACTION
Debug: [executoragent] AIRSExecutor SeqNo:  99ed
Debug: [executoragent] ExecutorAgent SeqNo :  a07d
Debug: [executoragent] Status     :  3
Debug: [executoragent] Version    :  14
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] Deploying Plugin: 2
Debug: [executoragent] User:
Debug: [executoragent] Aditional:
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
```

Tabla 40. Salida-Ejecución del plugin ipt-block para bloqueo de tráfico de entrada.

En la Figura 39 se puede corroborar el resultado de la ejecución de la respuesta. En virtud de que el objetivo de la respuesta es aislar al host atacante, se procede de la siguiente forma:

- En el router interno se bloquea todo el tráfico entrante del atacante, evitando que pueda efectuar un ataque a un host ubicado en otras subredes. No obstante, el atacante aún estaría en la capacidad de perpetrar un ataque en contra de un host perteneciente a la misma subred.

- Para aislar completamente al atacante, se agrega una regla de filtrado a cada host perteneciente a la misma subred del atacante. Ver Figura 39.

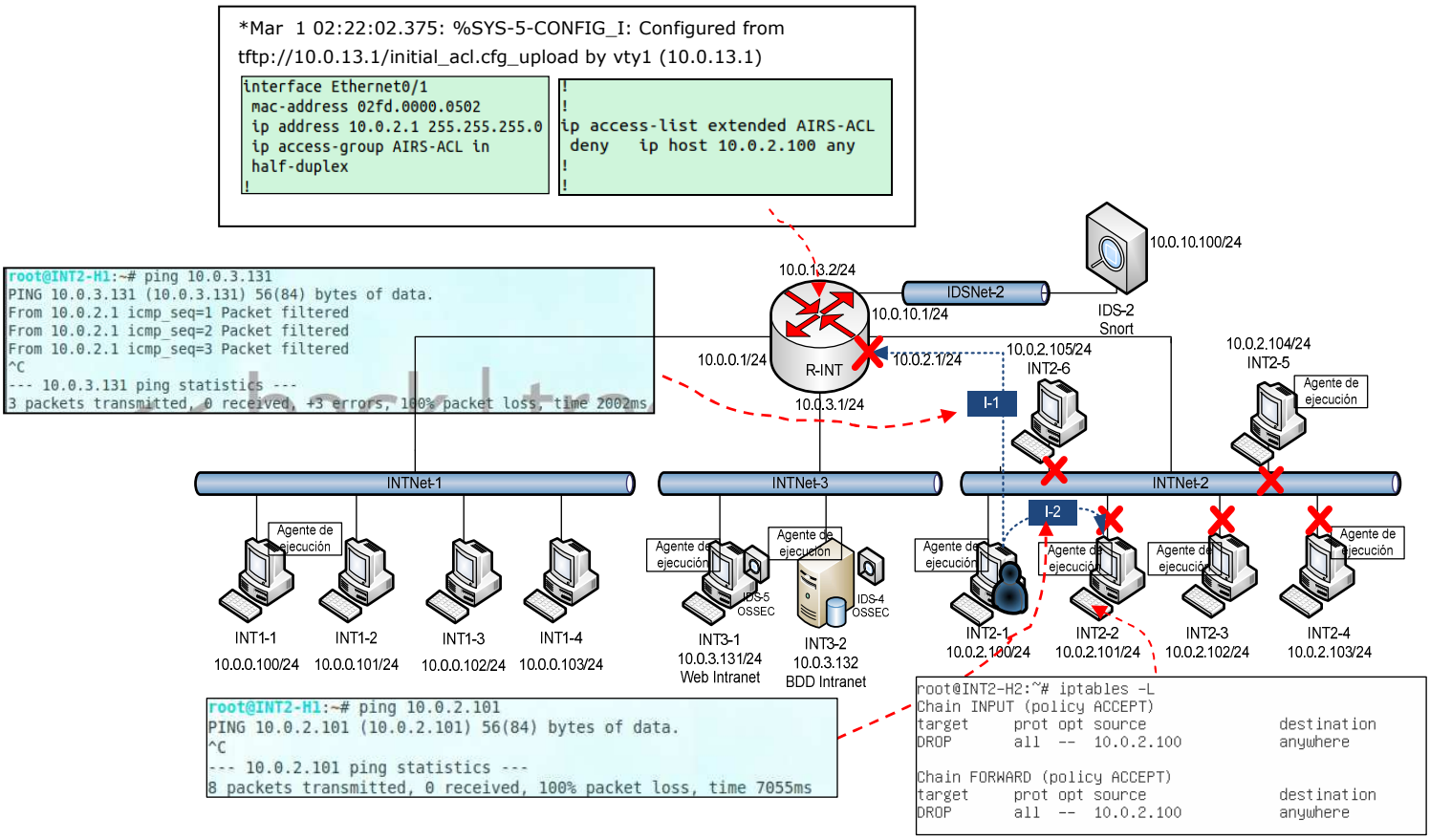



Figura 39. Resultado de la ejecución de la acción de respuesta.

## 7.4 Escenario 3: Respuesta activa compuesta a un ataque externo

Una de las características de la arquitectura propuesta es su capacidad para ejecutar acciones de respuesta interactuando con diferentes componentes de seguridad. Esto a su vez permite ejecutar un conjunto de acciones de distinto tipo; en esta sección el razonador infiere una respuesta compuesta que involucra un conjunto de 3 acciones: una respuesta pasiva, una respuesta activa de decepción y una respuesta de recuperación.

En la Figura 40, se observa una representación del escenario para la ejecución de la respuesta compuesta ante un ataque desde el exterior de la red de la organización.

### 7.4.1 Detección

- **E-1**: El atacante (10.1.200.22), intenta realizar un defacement por medio de un ataque web que utiliza la técnica de inyección SQL. Para ello se ha supuesto la modificación del fichero *index.html*, ubicado en el directorio */var/www/* del servidor web de la DMZ (192.168.100,130).

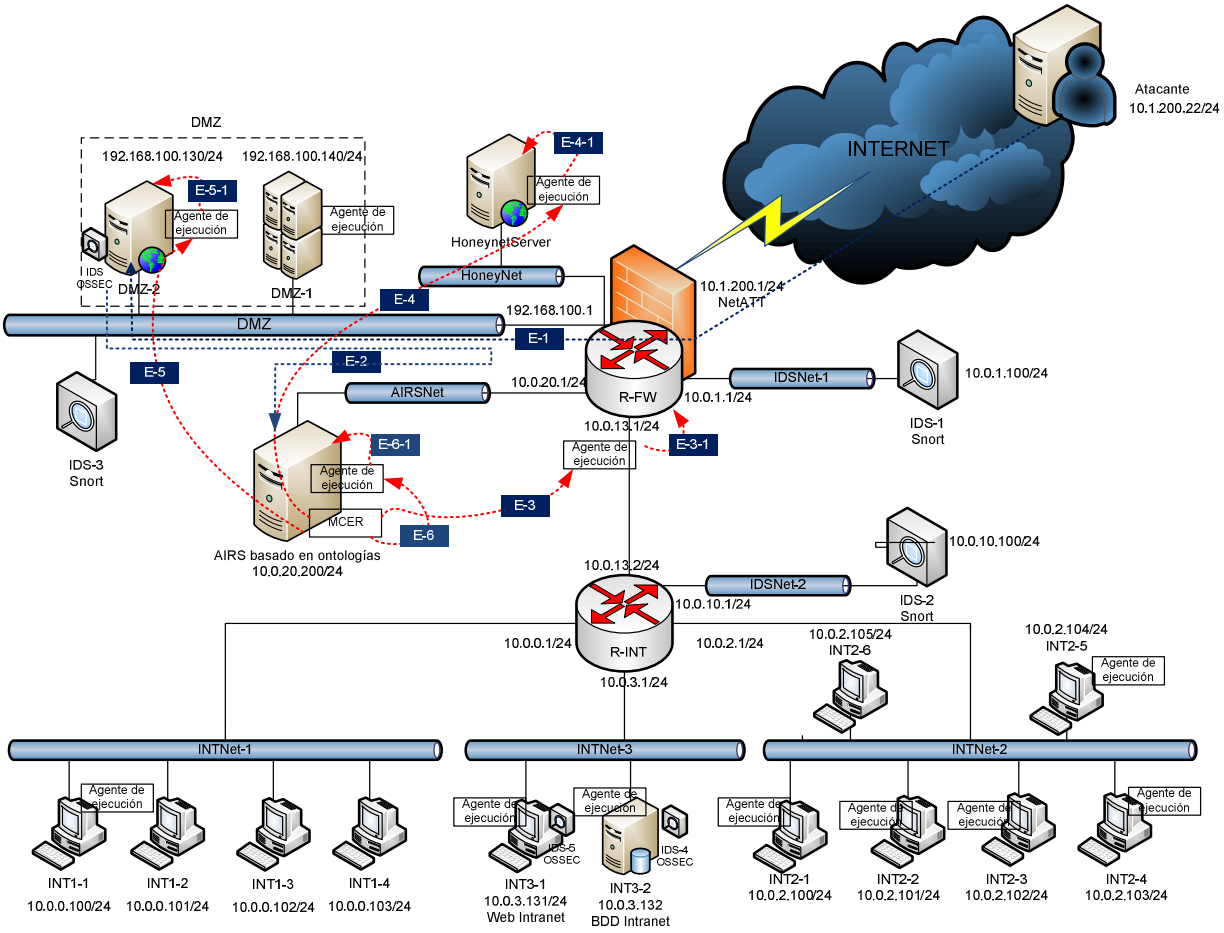



Figura 40. Respuesta compuesta a un ataque desde el exterior de la red.

- **E-2:** El servidor DMZ-2 incluye un HIDS OSSEC, a quién se le ha habilitado el sistema verificador de integridad (syscheckd). Syscheckd es un servicio que se ejecuta cada 30 minutos (configurable) y verifica la integridad de los ficheros contenidos en el directorio /var/www. Cuando detecta que el fichero *index.html* ha sido modificado (Ver Tabla 41) emite una alerta de intrusión en formato syslog hacia el razonador AIRS.

```
Received From: DMZ-2->syscheck
Rule: 551 fired (level 7) -> "Integrity checksum changed again (2nd time)."
Integrity checksum changed for: '/var/www/index.html'
Size changed from '4096' to '12332'
Old md5sum was: '195be3b887d824b5a4d4eb3c7865363f'
New md5sum is : '3f7249d1e8ceb81388f68d0202299780'
```

Tabla 41. Detección de la modificación del fichero index.html.

El razonador AIRS, escucha a través de su módulo *de recepción de alertas* en el puerto 512 y recibe las alertas en formato syslog.

### 7.4.2 Inferencia

Luego de la recepción de la alerta, el razonador AIRS infiere la respuesta óptima (ver Tabla 42). En este caso el razonador infiere una respuesta activa compuesta, cuyo nombre es *relayAttack* que tiene el propósito de redirigir al atacante hacia una honeynet

para analizar su modus operandi, restaurar el archivo modificado y notificar al administrador mediante un mensaje de texto. La respuesta inferida está compuesta de cuatro acciones simples: las acciones *honeynetDeployment* y *redirectTraffic* trabajan en conjunto para ejecutar una respuesta activa de decepción: la primera, despliega una red señuelo utilizando la herramienta de virtualización VNX, y la segunda redirige el tráfico del atacante hacia dicha red señuelo; la acción *restoreIndex,* ejecuta una respuesta activa de recuperación a través de la descarga del fichero *index.html* original desde un servidor FTP seguro y reemplaza al fichero modificado por el atacante; finalmente la acción *mailNotification*, ejecuta una respuesta pasiva a través del envío de un correo electrónico al administrador de la red.

Para la ejecución de la respuesta llama al módulo central de ejecución de respuestas (MCER), a quien provee el nombre de respuesta inferida (*relayAttack),* y los parámetros requeridos: la dirección IP del atacante (10.1.200.22) y la dirección IP del objetivo del ataque (*DMZ-1*).

**Syslog alert received.**
<132>Mar 7 19:50:10 DMZ-2 ossec: Alert Level: 7; Rule: 551 - Integrity checksum changed for /var/www/index.html.; Location: DMZ-2->/var/ossec/log/alerts.log; srcip: 10.1.200.22; Mar 7 19:50:10 DMZ-2 ...
**Inferring optimal response to a type 'Backdoor' intrusion...**
**Responses allowed**: blockInAttack, blockOutAttack, blockInOutAttack, addACL, denyConnectionHost, redirectTraffic, honeynetDeployment, relayAttack, mailNotification,
**Parameters received**--> HIDS:true, Attacker_IP:10.1.200.22, Victim_IP:DMZ-2, Attacker_port:null, Victim_port:null, Protocol:null, IntrusionType:Backdoor, SID:551
**Inferred response**: relayAttack

**Tabla 42. Salida-Inferencia de respuesta de óptima para un ataque de defacement.**

### 7.4.3 Ejecución

El MCER recibe la solicitud de respuesta desde el razonador AIRS y recupera toda la información necesaria para ejecutar la respuesta *relayAttack*. La información asociada a ésta respuesta se muestra en el siguiente cuadro:

Debug:[MCER] Parsing a new group of executor agents:**PerimetralFirewallGroup**
1.R-FW/10.0.20.1, Port:34000, Key:m1p455w0rd
Debug:[MCER] Parsing a new group of executor agents:**LocalAgentGroup**
1.local-agent/127.0.0.1, Port:898, Key:m1p455w0rd
Debug:[MCER] Parsing a new group of executor agents:**HoneyNetServerGroup**
1.HN-Server/127.0.0.1, Port:34001, Key:m1p455w0rd
Debug:[MCER] Parsing a new group of executor agents:**DMZServer**
1.DMZ-2/192.168.100.130, Port:34000, Key:m1p455w0rd

Debug:[MCER] Parsing a new response action:**relayAttack**, Type:**Composed**, Responses: **honeynetDeployment redirectTraffic restoreIndex mailNotification**
Debug:[MCER] Parsing a new response action:**honeynetDeployment**, Type:**Simple**, Group Executors Agents:**HoneyNetServerGroup**, Plugin:**hnvnx**, Who:**src**, Action:**execute**, Duration:**2hours**, Mode-Block:**0**, SIDs group:**ALL**
Debug:[MCER] Parsing a new response action:**redirectTraffic**, Type:**Simple**, Group Executors Agents:**PerimetralFirewallGroup**, Plugin:**ipt-redirect**, Who:**src**, Action:**execute**,

```
Duration:2hours, Mode-Block:conn, SIDs group:ALL
Debug:[MCER] Parsing a new response action:restoreIndex, Type:Simple, Group Executors Agents:DMZServer, Plugin:download-backup-ftp, Who:src, Action:execute, Duration:0, Mode-Block:0, SIDs group:ALL
Debug:[MCER] Parsing a new response action:mailNotification, Type:Simple, Group Executors Agents:LocalAgentGroup, Plugin:email, Who:src, Action:execute, Duration:0, Mode-Block:in, SIDs group:ALL
```

Dicha información señala que la respuesta *relayAttack* está compuesta por 4 acciones: *honeynetDeployment, redirectTraffic, restoreIndex y mailNotification.*

- La acción *honeynetDeployment,* se ejecuta sobre el agente de ejecución a través del plugin *hnvnx* para desplegar la honeynet.
- La acción redirectTraffic, se ejecuta a través del plugin *ipt-redirect* sobre el agente de ejecución del firewall perimetral y redirige todo el tráfico hacia la honeynet desplegada.
- La acción *restoreIndex,* emplea el plugin *download-backup-ftp* para reemplazar el fichero */var/www/index.html* modificado por el fichero original descargado desde un servidor FTP.
- Finalmente, la acción *mailNotificación* envía una notificación mediante un correo electrónico al administrador; para ello hace uso del plugin *email.*

La respuesta inferida involucra a 4 agentes de ejecución, el MCER entonces a través del módulo de comunicación establece 4 conexiones para emitir las solicitudes de respuesta correspondientes. Ver Tabla 43.

```
Debug:[MCER] Selected response action:blockInAttack
[7/2/2013 1:0:51] [3] [MCER] Info:[Main] Trying to send 'relayAttack' response action request to communication module...
Info: [CheckIn] Connected to host 10.0.20.1.
Info: [ExecuteResponseAction] Connected to host 10.0.20.1. Executing action IP 10.1.200.22.
Info: [CheckOut] Disconnecting from host 10.0.20.1.
Info: [CheckIn] Connected to host 127.0.0.1.
Info: [ExecuteResponseAction] Connected to host 127.0.0.1. Executing action IP 10.1.200.22.
Info: [CheckOut] Disconnecting from host 10.0.20.1.
        [...Líneas omitidas...]
Info: [CheckIn] Connected to host 192.168.100.130.
Info: [ExecuteResponseAction] Connected to host 192.168.100.130. Executing action IP 10.1.200.22.
Info: [CheckOut] Disconnecting from host 192.168.100.130.
_____:::::::::::::::::::::::_____-- esperando alertas nuevas
```

**Tabla 43. Salida-Envío de solicitudes a los agentes de ejecución a través del módulo de comunicación.**

Antes de la ejecución de las acciones de respuesta, los plugins involucrados realizan el parsing de los parámetros que requieren para su ejecución. El plugin *hnvnx* obtiene: la ruta del fichero VNX en donde se define el escenario a desplegar y el nombre de usuario; el plugin *ipt-redirect* obtiene la dirección IP del host a

donde redirigirá el tráfico; el plugin *download-backup-ftp* obtiene la dirección IP, usuario y contraseña del servidor FTP, el fichero a restaurar y el directorio en donde se descarga; el plugin *email* obtiene la dirección del servidor SMTP, la dirección de correo electrónico del emisor y destinatario:

| **Sobre el agente de ejecución local (VNX):** |
|---|
| Info: [executoragent] Linking plugin 'hnvnx'...<br>Info: [executoragent] Initializing plugin 'hnvnx'...<br>Debug: [hnvnx] Plugin Parsing...<br>Debug: [hnvnx] Adding HNVNX Parameters: file "/usr/share/vnx/examples/honeynet1.xml", user "root" |
| **Sobre el agente de ejecución local (EMAIL):** |
| Info: [executoragent] Linking plugin 'email'...<br>Debug: [email] Plugin Parsing...<br>Debug:[email] Adding email plugin parameters: SMTP Server: 127.0.0.1, Recipient: danny.guamanl@gmail.com, Sender: admin@upm.es |
| **Sobre el agente de ejecución DMZ-2:** |
| Info: [executoragent] Linking plugin 'download-backup-ftp'...<br>Info: [executoragent] Initializing plugin 'download-backup-ftp'...<br>Debug: [updownftp] Plugin Parsing...<br>Debug: [host-deny] Adding UpDownFTP Parameters: Server:192.168.100.1, Username:santiago, Password<br>ntiago, File Path Source:./index.html, File Path Destination:/var/www/ |

- **E3 y E3-1:** El agente de ejecución a través del plugin *ipt-redirect* agrega dos reglas al firewall perimetral IPTables. La primera, agrega una regla a la cadena FORWARD de la tabla FILTER, de tal forma que deseche todo el tráfico proveniente del atacante (10.1.200.22). La segunda, agrega una regla a la cadena PREROUTING de la tabla NAT para redirigir el tráfico proveniente del atacante (10.1.200.22) hacia el host 192.168.100.150. En la Tabla 44 se puede observar la salida de la ejecución del plugin ipt-redirect.

Debug: [executoragent] Connection from: 10.0.20.200
[…Líneas omitidas…]
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] **Deploying Plugin: 7**
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] **Executing response action on host 10.1.200.22 inbound for 7200 seconds (Sig_ID: 0).**
Debug: [ipt-redirect] **Executing plugin...**
Debug: [ipt-redirect] **Command /sbin/iptables -A FORWARD -s 10.1.200.22 -j DROP**
Debug: [ipt-redirect**] Command2 /sbin/iptables -t nat -A PREROUTING -s 10.1.200.22 -j DNAT --to-destination 192.168.100.150**
Debug: [executoragent] Connection from: 127.0.0.1.
Debug: [executoragent] Received Packet: CHECKOUT

**Tabla 44. Salida-Ejecución del plugin ipt-redirect para redirección del tráfico de entrada del atacante.**

- **E4 y E4-1:** El agente de ejecución local ejecuta la segunda acción a través del plugin *hnvnx*. La red señuelo virtual se despliega utilizando la herramienta de

virtualización Virtual Network over LinuX (VNX). Previo a la ejecución de esta acción, el administrador debe crear el fichero VNX con el escenario virtual a desplegar. Ver Tabla 45.

```
Debug: [executoragent] Adding sensor 127.0.0.1 to list.
Debug: [executoragent] Received Packet: CHECKIN
Debug: [executoragent] AIRSExecutor SeqNo:  6f8e
Debug: [executoragent] ExecutorAgent SeqNo :  4350
Debug: [executoragent] Status     :  1
Debug: [executoragent] Version    :  15
Debug: [executoragent] Connection from: 127.0.0.1.
        [...Lineas omitidas...]
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] Deploying Plugin: 3
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] Executing response action on host 10.1.200.22 inbound for 7200 seconds (Sig_ID: 0).
Debug: [hnvnx][7f415971b700] Plugin being executed...
Debug: [hnvnx][7f415971b700] command->  vnx -f /usr/share/vnx/examples/honeynet1.xml -t -v -u root
Debug: [executoragent] Connection from: 127.0.0.1.
Debug: [executoragent] Received Packet: CHECKOUT
```

Tabla 45. Salida-Ejecución del plugin hnvnx para despliegue de la honeynet a través de VNX.

- **E5 y E5-1:** El agente de ejecución ubicado en el servidor DMZ-2 (192.168.100.130) a través del plugin *download-backup-ftp* restaura el fichero *index.html* desde un servidor FTP seguro.

```
Debug: [executoragent] Connection from: 192.168.100.1.
Tamaño del paquete recibido: 86
Tamaño a recibir: 86
Debug: [executoragent] Received Packet: RESPONSEACTION
Debug: [executoragent] AIRSExecutor SeqNo:  9ee0
Debug: [executoragent] ExecutorAgent SeqNo :  5d63
Debug: [executoragent] Status     :  3
Debug: [executoragent] Version    :  14
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] Deploying Plugin: 16
Debug: [executoragent] User:
Debug: [executoragent] Aditional:
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] Executing response action on host 10.1.200.22 inbound for 0 seconds (Sig_ID: 0
).
Debug: [updownftp] Plugin will be executed...
Debug: [updownftp] Trying to establish connection to ftp server at 192.168.100.1
Debug: [updownftp] Connected to ftp server at 192.168.100.1.
Debug: [executoragent] Connection from: 192.168.100.1.
Tamaño del paquete recibido: 86
Tamaño a recibir: 86
Debug: [executoragent] Received Packet: CHECKOUT
Debug: [executoragent] AIRSExecutor SeqNo:  9b23
Debug: [executoragent] ExecutorAgent SeqNo :  fc43
Debug: [executoragent] Status     :  2
Debug: [executoragent] Version    :  14
Debug: [updownftp] User santiago has been authenticated to ftp server at 192.168.100.1
Debug: [updownftp] /var/www/index.html-070313214248 file backup created!
Debug: [updownftp] Downloading ./index.html file backup to /var/www/index.html from ftp server at 19
2.168.100.1!
Debug: [updownftp] Successful execution!
```

Tabla 46. Salida-Ejecución del plugin download-backup-ftp para restauración del fichero index.html desde un servidor FTP.

Como se observa en la Tabla 46, la ejecución de este plugin incluyen las siguientes tareas:

- Establece una conexión con el servidor FTP y posteriormente se autentica.
- Realiza un backup del fichero modificado para su análisis posterior.
- Finalmente descarga el fichero *index.html* desde el servidor FTP y lo reemplaza por el fichero modificado.

- **E6 y E6-1:** El agente de ejecución local a través del plugin *email,* envía una notificación envía un correo electrónico al destinatario que fue definido como parámetro. Para ello, establece una conexión con el servidor SMTP que actúa como relay para emitir el correo electrónico. Los comandos emitidos se puede observar en la Tabla 47.

```
Starting to listen for AIRSExecutor alerts
Debug: [executoragent] Connection from: 127.0.0.1.
Debug: [executoragent] Adding sensor 127.0.0.1 to list.
Debug: [executoragent] Received Packet: CHECKIN
Debug: [executoragent] AIRSExecutor SeqNo:  b5d4
Debug: [executoragent] ExecutorAgent SeqNo :  413e
Debug: [executoragent] Status     :  1
Debug: [executoragent] Version    :  15
Debug: [executoragent] Connection from: 127.0.0.1.
        [...Líneas omitidas...]
Debug: [executoragent] Execute-Action request received...
Debug: [executoragent] Deploying Plugin: 4
Debug: [executoragent] ResponseAction triggered by Signature ID: 0
Info: [executoragent] Executing response action on host 10.1.200.22 inbound for 0 seconds (Sig_ID: 0).
Debug: [email] Plugin Sending Mail...
Debug: [email] Connected to mail server at 127.0.0.1.
Debug: [email] Waiting from 127.0.0.1 for: 220
Debug: [email] Sending 17 bytes to 127.0.0.1: HELO danny-host
Debug: [email] Waiting from 127.0.0.1 for: 250
Debug: [email] Sending 6 bytes to 127.0.0.1: RSET
        [...Líneas omitidas...]
Debug: [email] Sending 6 bytes to 127.0.0.1: DATA
Debug: [email] Waiting from 127.0.0.1 for: 354
Debug: [email] Sending 305 bytes to 127.0.0.1: From: admin@upm.es To: danny.guamanl@gmail.com Subject: Intrusion detected in host 10.1.200.22 Date: Thu, 07 Mar 2013 21:39:29 +0100 Based Ontologies AIRS has detected an intrusion in connection 10.1.200.22->192.168.100.130:1 (tcp) This intrusion detection was triggered by signature ID: 0 .
Debug: [email] Waiting from 127.0.0.1 for: 250
Debug: [email] Sending 6 bytes to 127.0.0.1: QUIT
Debug: [email] Email has been sent. Now waiting 5 secs...
Debug: [executoragent] Connection from: 127.0.0.1.
Debug: [executoragent] Received Packet: CHECKOUT
```

**Tabla 47. Salida-Ejecución del plugin email para envío de un correo electrónico a través de un relay SMTP.**

En la Figura 41 se puede ratificar el resultado de la ejecución de la respuesta compuesta:

- **Respuesta de recuperación**: El fichero *index.html* modificado es restaurado a su estado original.
- **Respuesta de decepción**: Se despliega una red señuelo virtual utilizando la herramienta VNX a la cual se redirige el tráfico del atacante, mediante la adición de una regla al firewall IPTables.
- **Respuesta pasiva**: Se envía un correo electrónico al administrador de la red.

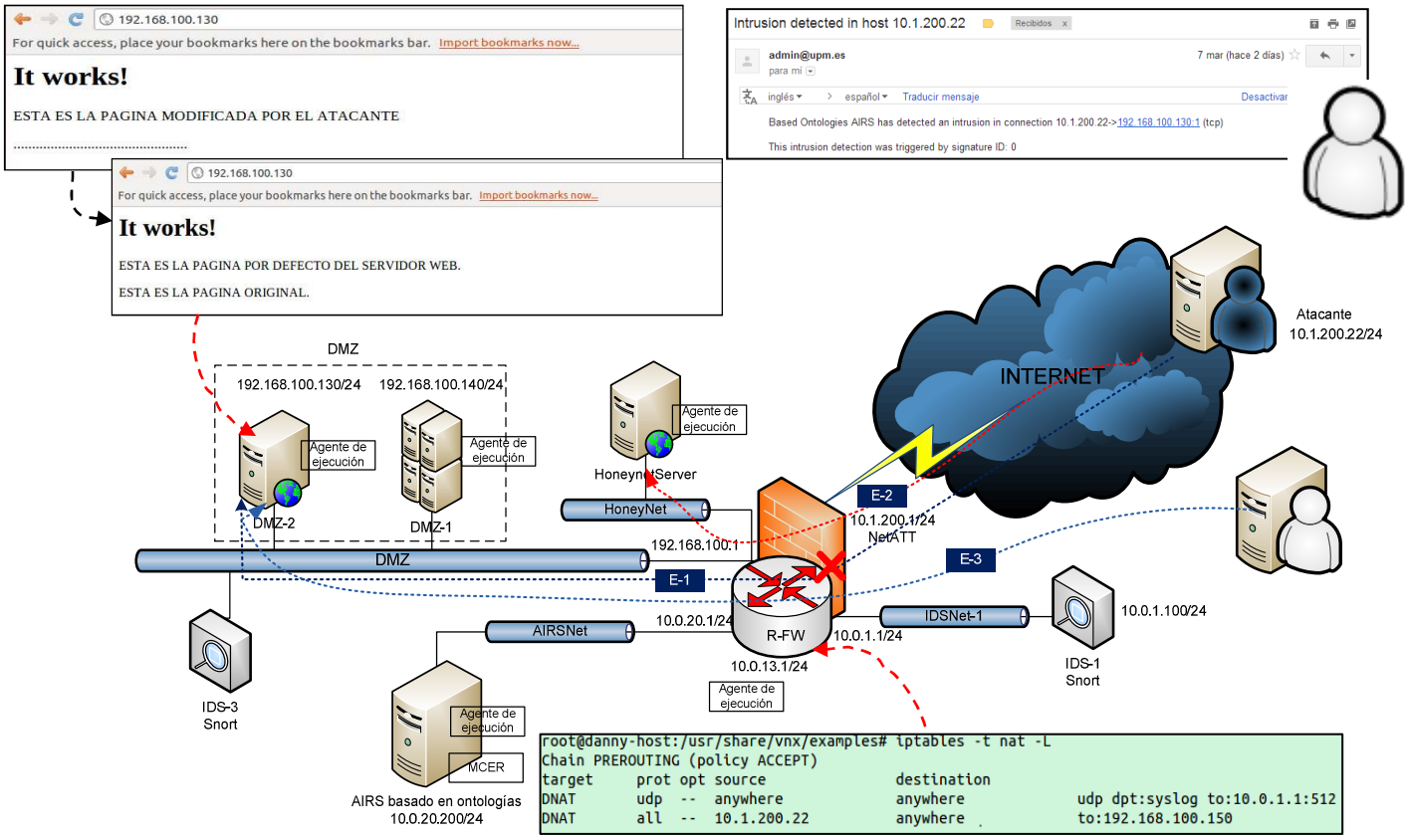



Figura 41. Resultado de la ejecución de la respuesta compuesta: decepción, recuperación y pasiva.

## 7.5 Análisis de resultados

En el capítulo 3 se determinaron tres problemas del módulo de ejecución de respuestas del AIRS basado en ontologías: primero, únicamente permitía la ejecución local de acciones de respuesta; segundo, tenía escalabilidad limitada dificultando el despliegue de nuevas acciones de respuesta, y; tercero, el catálogo de acciones de respuestas era muy reducido.

En virtud de ello, y como se puede corroborar en los tres escenarios propuestos, el presente trabajo de fin de máster aborda la solución a dichos problemas desde dos vértices:

1. A través de la propuesta de una arquitectura del ejecutor de respuestas distribuida, segura y escalable, cuyos requerimientos funcionales y no funcionales fueron establecidos en las Tablas 19 y 20 del capítulo 4 respectivamente ,y;
2. A través de la provisión de pruebas de concepto de acciones de respuesta para el catálogo del AIRS, que aprovechen la naturaleza distribuida de la arquitectura propuesta.

En los tres escenarios se puede ratificar que la provisión de agentes de ejecución distribuidos, permite la ejecución de respuestas de forma local y remota, interactuando con diversos componentes de seguridad: firewalls IPTables, VNX, routers cisco, servidor ftp, servidor de correo, tcpwrapper, gestor de usuarios de Linux, entre otros.

De la misma manera en los tres escenarios existe un marco común de comunicación y un conjunto de servicios "extras" independientemente de la respuesta a ejecutar. El marco común de comunicación permite establecer conexiones seguras y confiables entre el MCER y los agentes de ejecución. Mientras que la lógica de ejecución de la respuesta, parsing de parámetros de configuración e inicialización, propios de cada componente de seguridad, son provistos a través de plugins.

En el escenario 1 se presenta una respuesta activa de protección simple que añade una regla de bloqueo al firewall perimetral IPTables. La mayoría de herramientas existentes proveen respuestas activas de este tipo; no obstante, aunque la ejecución de respuestas sobre un firewall perimetral es considerablemente efectiva, tiene algunas limitaciones:

- Se asume que todo el tráfico de entrada y salida atraviesa dicho dispositivo; sin embargo, existe gran facilidad para establecer nuevos puntos de entrada y salida a través de dispositivos móviles o redes ad hoc inalámbricas, convirtiéndose en puertas de acceso por medio de las cuales se puede perpetrar un ataque.
- Además, se asume que el perímetro interno es de confianza; sin embargo los ataques pueden ser originados desde el interior de la red.

Cómo se puede apreciar en el escenario 2, la arquitectura distribuida permite solventar tales inconvenientes a través de la interacción con firewalls basados en host (personales). Además cabe recalcar las siguientes ventajas:

- Aunque el ataque originalmente es dirigido hacia un servidor de otra subred, se ejecuta una respuesta preventiva mediante el bloqueo sobre los firewalls de los hosts pertenecientes a la misma subred del atacante.
- Se puede realizar el bloqueo empleando otros plugins para proveer varias capas de seguridad. Por ejemplo, como complemento al bloqueo sobre el firewall IPTables, se puede agregar la dirección IP del atacante al fichero host.deny, empleado por TCPWrapper para filtrar el acceso hacia ciertos servicios basados en TCP/IP. El plugin *host-deny,* que ha sido añadido al catálogo del AIRS, provee esta funcionalidad.

Otra de las características importantes a valorar se puede apreciar en el escenario 3, en donde se despliega una respuesta compuesta por 3 tipos de acciones: decepción, recuperación y una respuesta pasiva. La arquitectura propuesta, al estar basada en plugins, permite un fácil despliegue de nuevas acciones de respuesta utilizando un marco común de comunicación y ciertas funcionalidades extras provistas por SnortSam. Nuevas acciones que interactúen con otros componentes de seguridad pueden ser agregados a la arquitectura de forma conectable.

Finalmente mencionar la importancia de contar con IDSs distribuidos y de varios tipos; en los tres escenarios se han empleado NIDS Snort y HIDS OSSEC. La utilización de múltiples IDSs permite detectar un rango más amplio de intrusiones; además, si varios IDSs cubren un mismo conjunto de intrusiones, la confianza sobre las alertas emitidas desde estos IDSs aumenta; reduciendo los falsos positivos.

# 8 Conclusiones y líneas de trabajo futuras

## 8.1 Conclusiones

Dado que el campo de la seguridad en redes ha requerido de una constante investigación para evaluar y optimizar los mecanismos de control de acceso y protección de la información en tránsito, que permitan mitigar los ataques; surgen los Sistemas de Respuesta a Intrusiones (IRSs), como nuevos mecanismos de defensa que permiten detectar y reaccionar ante intrusiones que atentan contra la integridad, confidencialidad y disponibilidad de un recurso. Los IRSs tiene por objetivo proveer un catálogo con varias acciones de respuesta, y la elección de una de ellas dependerá de un análisis previo que valore el costo que supone una intrusión respecto del costo de ejecutar una acción de respuesta.

Dentro de las diferentes investigaciones que se llevan cabo en torno a los IRS, se encuentra el AIRS basado en ontologías; un proyecto de investigación emprendido en la Universidad Politécnica de Madrid y en la que precisamente se enmarca el presente trabajo de fin de máster.

Luego de realizar una revisión del estado del arte y de ciertos conceptos necesarios para comprender el entorno de trabajo, se procedió a evaluar el estado actual del AIRS basado en ontologías enfocándonos en el módulo ejecutor de respuestas. Al finalizar la evaluación, se determinaron tres problemas de dicho módulo:

1. El módulo permitía únicamente la ejecución local de acciones de respuesta; es decir, en el mismo host sobre el que se ejecuta el AIRS. No obstante, en el estado del arte se pudo determinar que una de las características fundamentales de un IRS autónomo es su capacidad para ejecutar acciones de respuestas interactuando con otros componentes de seguridad locales y remotos.
2. El módulo no es escalable, ya que ante la carencia de una arquitectura se dificultaba el despliegue de nuevas acciones de respuesta.
3. El módulo contiene un catálogo de acciones de respuestas muy reducido. Este problema también era consecuencia de la carencia de una arquitectura escalable, que permita el rápido y simple despliegue de nuevas acciones de respuesta, que a su vez dificulta la validación del AIRS basado en ontologías.

Como solución a los problemas antes mencionados se efectuaron dos tareas:

1. La propuesta de una arquitectura del ejecutor de respuestas, que sea distribuida, segura y escalable. Para ello se realizó un análisis de requerimientos, diseño y posterior implementación de un prototipo, que finalmente fue integrado al AIRS basado en ontologías.
2. La implementación de pruebas de concepto de acciones de respuesta para el catálogo del AIRS, que aprovechan las características de la arquitectura propuesta.

Como decisión en la fase de diseño se procedió a reutilizar algunos componentes de la aplicación open source SnortSam. SnortSam funciona como un plugin de salida del NIDS Snort y ejecuta acciones de bloqueo sobre diferentes firewalls. Se aprovechó entonces algunas funcionalidades extras, que van más allá de la ejecución autónoma de acciones de bloqueo. El módulo de comunicación y parte de la funcionalidad de los agentes de ejecución han sido reutilizados.

La arquitectura del ejecutor de respuestas propuesto, involucra a 6 componentes: *los IDSs*, se trabajó con NIDS Snort y HIDS OSSEC quienes se encargan de la detección de las intrusiones; *el razonador AIRS,* infiere la respuesta óptima a través de la evaluación de varias métricas de respuesta; *el MCER,* el módulo central de ejecución se encarga de construir una solicitud de respuesta y ubicar a los agentes de ejecución; *el módulo de comunicación,* establece una conexión confiable y segura entre el MCER y los agentes de ejecución; *los agentes de ejecución,* ejecutan funciones de autenticación, listas blancas de ejecución, y la ejecución misma de la acción de respuesta; *el componente de seguridad,* que representa el dispositivo que lleva a cabo la acción de respuesta real utilizando su interfaz de línea de comandos, tal que se altere su funcionamiento tan pronto como es ejecutado.

La utilización de múltiples IDSs distribuidos y de varios tipos (en este caso HIDS y NIDS), permite detectar un rango más amplio de intrusiones. Además que si varios IDSs cubren un mismo conjunto de intrusiones, la confianza sobre las alertas emitidas desde estos IDSs aumenta; por consiguiente se reducen los falsos positivos.

La provisión de agentes de ejecución distribuidos permite la ejecución de respuestas de forma local y remota, interactuando con diversos componentes de seguridad. Así mismo provee un marco común de comunicación y un conjunto de servicios independientemente de la respuesta a ejecutar. El marco común de comunicación permite establecer conexiones seguras y confiables entre el MCER y los agentes de ejecución. Mientras que la lógica de ejecución de la respuesta, parsing de parámetros de configuración e inicialización, propios de cada componente de seguridad, son provistos a través de plugins.

La arquitectura propuesta, al estar basada en plugins, permite un fácil despliegue de nuevas acciones de respuesta utilizando, como se mencionó anteriormente, un marco común de comunicación. Nuevas acciones de respuesta (protección, decepción, recuperación y pasivas) que interactúen con otros componentes de seguridad pueden ser agregados a la arquitectura de forma conectable.

Para validar el ejecutor de respuestas, se ha integrado al AIRS basado en ontologías y se ha desplegado una red de pruebas utilizando la herramienta VNX. Se proponen 3 escenarios sobre los cuales se despliegan diferentes respuestas ante ataques originados desde el interior y exterior de la red e la organización, obteniéndose resultados satisfactorios.

## 8.2 Líneas de trabajo futuras

Una vez provisto e implemento el ejecutor de respuestas, nuevos plugins pueden ser implementados. Los plugins provistos en este trabajo son pruebas de concepto que pueden ser mejorados; a través de la implementación de plugins más "inteligentes" la obtención de ciertos parámetros se pueden obtener de forma dinámica.

La naturaleza basada en plugins, permite definir nuevas funciones que deben ser implementadas por cada plugin. Actualmente, no existe una función que evalúe el efecto que tuvo la acción de respuesta; es decir, si la acción neutralizó o no el ataque. Entonces, una función para efectuar dicha evaluación puede ser muy útil para que el razonador AIRS utilice el resultado de éxito de una acción de respuesta en inferencias futuras.

Actualmente, cuando se requiere ejecutar una respuesta compuesta, es el MCER quien debe ejecutar cada acción simple de forma independiente; no obstante, en determinadas circunstancias podría ser útil que un agente de ejecución tenga la capacidad de ejecutar una acción sobre otro agente de ejecución. Esta funcionalidad podría implementarse como un plugin adicional, para evitar modificar la arquitectura original.

Una arquitectura basada en plugins similar a la propuesta en esta memoria, podría ser extendida hacia los IDSs. Es decir, proveer un marco común de comunicación entre los IDSs y el receptor de alertas, e implementar un plugin que incluya una función para realizar el parsing de cada alerta.

# 9 Bibliografía


[1] Symantec. Symantec internet security threat report, 2011 trends. Symantec Corporation. USA. 2012Available: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf.

[2] J. P. Anderson, "Computer security threat monitoring and surveillance," *National Institute of Standards and Technology (NIST),* 1980.

[3] N. B. Anuar, M. Papadaki, S. Furnell and N. Clarke, "An investigation and survey of response options for Intrusion Response Systems (IRSs)," *Information Security for South Africa (ISSA), 2010,* pp. 1-8, 2010.

[4] M. Papadaki and S. Furnell, "IDS or IPS: what is best?" *Network Security,* vol. 2004, pp. 15-19, 2004.

[5] N. Stakhanova, S. Basu, J. Wong, D. P. IEEE Tech Comm and Nokia, "A cost-sensitive model for preemptive intrusion response system," pp. 9, 2007.

[6] M. Tavallaee, N. Stakhanova and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 40, pp. 516-524, 2010.

[7] G. M. Nazer and A. A. L. Selvakumar, "Current Intrusion Detection Techniques in Information Technology–A Detailed Analysis," *European Journal of Scientific Research,* vol. 65, pp. 611-624, 2011.

[8] S. Mallissery, J. Prabhu and R. Ganiga, "Survey on intrusion detection methods," in *Advances in Recent Technologies in Communication and Computing (ARTCom 2011), 3rd International Conference on,* 2011, pp. 224-228.

[9] N. Stakhanova, S. Basu and J. Wong, "A taxonomy of intrusion response systems," *International Journal of Information and Computer Security,* vol. 1, pp. 169-184, 2007.

[10] V. Mateos, V. Villagrá and F. Romero, "Ontologies-Based Automated Intrusion Response System," *Computational Intelligence in Security for Information Systems 2010,* pp. 99-106, 2010.

[11] V. Mateos, V. A. Villagrá, F. Romero and J. Berrocal, "Definition of response metrics for an ontology-based Automated Intrusion Response Systems," *Comput. Electr. Eng.,* 2012.

[12] M. Wood and M. Erlinger, "Intrusion detection message exchange requirements," *RFC 4766,* 2007.

[13] OSSEC. OSSEC v2.7.0 documentation. log samples. *2013(01/03),* 2012. Available: http://www.ossec.net/doc/log_samples/index.html.

[14] Snort. **SNORT users manual 2.9.3**. *2013(01/03),* pp. 244. 2012. Available: http://www.snort.org/assets/166/snort_manual.pdf.

[15] H. Debar, D. A. Curry and B. S. Feinstein, "The intrusion detection message exchange format (IDMEF)," 2007.

[16] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar and M. Dagenais, "Intrusion response systems: survey and taxonomy," *Int J Comput Sci Network Secur (IJCSNS),* vol. 12, pp. 1-14, 2012.

[17] S. H. Amer and J. A. Hamilton Jr, "Intrusion Detection Systems (IDS) Taxonomy-A Short Review," *Defense Cyber Security,* vol. 13, 2011.

[18] V. Villagrá, *Seguridad En Redes De Telecomunicación.* Madrid, Spain: ISDEFE, 2009.

[19] P. de Boer and M. Pels, "Host-based intrusion detection systems," *Amsterdam University,* 2005.

[20] D. G. Gómez, "Sistemas de Detección de Intrusiones," *Sistemas De Detección De Intrusiones,* 2003.

[21] B. Mukherjee, L. T. Heberlein and K. N. Levitt, "Network intrusion detection," *Network, IEEE,* vol. 8, pp. 26-41, 1994.

[22] Y. Shi, Y. Tian, G. Kou, Y. Peng and J. Li, "Network Intrusion Detection," *Optimization Based Data Mining: Theory and Applications,* pp. 237-241, 2011.

[23] D. Gonzales-Gómez. Network taps. *2013(01/04),* 2005. Available: http://www.dgonzalez.net/papers/roc_es/node4.html.

[24] M. Gyanchandani, J. Rana and R. Yadav, "Taxonomy of Anomaly Based Intrusion Detection System: A Review," *Neural Networks,* vol. 42, pp. 44, 2012.

[25] D. Bolzoni, *Revisiting Anomaly-Based Network Intrusion Detection Systems.* University of Twente, 2009.

[26] D. J. Ragsdale, C. A. Carver Jr, J. W. Humphries and U. W. Pooch, "Adaptation techniques for intrusion detection and intrusion response systems," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on,* 2000, pp. 2344-2349.

[27] M. A. Aydın, A. H. Zaim and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Comput. Electr. Eng.,* vol. 35, pp. 517-526, 5, 2009.

[28] J. Aussibal and L. Gallon, "A new distributed IDS based on CVSS framework," *Web-Based Information Technologies and Distributed Systems,* pp. 189-206, 2010.

[29] G. Vigna and R. A. Kemmerer, "NetSTAT: A network-based intrusion detection approach," in *Computer Security Applications Conference, 1998. Proceedings. 14th Annual,* 1998, pp. 25-34.

[30] E. Peter and T. Schiller, "A Practical Guide to Honeypots," *Washington Univerity,* 2011.

[31] L. Spitzner, "Know your enemy: Honeynets," *Honeynet Project,* 2005.

[32] A. Hoskins, Y. Liu and A. Relkuntwar, "Counter-Attacks for Cybersecurity Threats," 7/12/2005, 2005.

[33] N. Desai. **Intrusion prevention systems: The next step in the evolution of IDS**. 2003. Available: http://www.symantec.com/connect/articles/intrusion-prevention-systems-next-step-evolution-ids.

[34] M. B. Rash, *Intrusion Prevention and Active Response: Deploying Network and Host IPS.* Syngress Media Incorporated, 2005.

[35] B. Foo, Y. S. Wu, Y. C. Mao, S. Bagchi and E. Spafford, "ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on,* 2005, pp. 508-517.

[36] C. Mu and Y. Li, "An intrusion response decision-making model based on hierarchical task network planning," *Expert Syst. Appl.,* vol. 37, pp. 2465-2472, 2010.

[37] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens and S. Dubus, "Risk-aware framework for activating and deactivating policy-based response," in *Network and System Security (NSS), 2010 4th International Conference on,* 2010, pp. 207-215.

[38] N. Kheir, N. Cuppens-Boulahia, F. Cuppens and H. Debar, "A service dependency model for cost-sensitive intrusion response," *Computer Security–ESORICS 2010,* pp. 626-642, 2010.

[39] M. Locasto, K. Wang, A. Keromytis and S. Stolfo, "Flips: Hybrid adaptive intrusion prevention," in *Recent Advances in Intrusion Detection,* 2006, pp. 82-101.

[40] C. A. Carver. Adaptative agent-based intrusion response. 2003.

[41] G. Stoneburner, A. Goguen and A. Feringa, "Risk management guide for information technology systems," *Nist Special Publication,* vol. 800, pp. 800-830, 2002.

[42] International Standard Organization, "ISO/IEC 27005, Information Security Risk Management," *2008,* .

[43] A. Årnes, K. Sallhammar, K. Haslum, T. Brekne, M. Moe and S. Knapskog, "Real-time risk assessment with network sensors and intrusion detection systems," *Computational Intelligence and Security,* pp. 388-397, 2005.

[44] M. Jahnke, C. Thul and P. Martini, "Graph based metrics for intrusion response measures in computer networks," in *Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on,* 2007, pp. 1035-1042.

[45] E. E. Schultz, "A framework for understanding and predicting insider attacks," *Comput. Secur.,* vol. 21, pp. 526-531, 2002.

[46] S. Hansman and R. Hunt, "A taxonomy of network and computer attacks," *Comput. Secur.,* vol. 24, pp. 31-43, 2005.

[47] V. Villagrá and V. Mateos, "Acciones de respuesta," *Telefónica Investigación y Desarrollo S.A.U. Universidad Politécnica De Madrid. Proyecto Cenit Segur@,* 2009.

[48] CVE Project. The standard for information security and vulnerability names. [Availability]. *2013(02/01),* 2013. Available: http://cve.mitre.org.

[49] R. Shirey, "RFC 2828: Internet Security Glosary," *RFC 2828,* 2000.

[50] W3C. OWL 2 web ontology language. *W3C Recommendation* 2012. Available: OWL 2 Web Ontology Language.

[51] W3C. SWRL: A semantic web rule language. 2004. Available: http://www.w3.org/Submission/SWRL/.

[52] M. Jang. Bossam Rule/OWL reasoner. *2013(12/01/2013),* Available: http://bossam.wordpress.com/about-bossam/.

[53] I. Sommerville and G. Kotonya, *Requirements Engineering: Processes and Techniques.* John Wiley & Sons, Inc., 1998.

[54] J. L. Thames, R. Abler and D. Keeling, "A distributed firewall and active response architecture providing preemptive protection," in *Proceedings of the 46th Annual Southeast Regional Conference on XX,* 2008, pp. 220-225.

[55] F. Knobbe. SnortSam. *A Firewall Blocking Agent for Snort 2013(02/01),* 2012. Available: A Firewall Blocking Agent for Snor.

[56] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall and N. Ferguson, "Twofish: A 128-bit block cipher," *NIST AES Proposal,* vol. 15, 1998.

[57] M. Roesch. Snort users manual. *2013(02/12),* 2012. Available: http://manual.snort.org/.

[58] OSSEC. Users manual OSSEC. *2013(02/12),* 2010. Available: http://www.ossec.net/doc/manual/index.html.