Centro de Identificación y Seguridad Digital

Facultad 1

Módulo de gestión de reglas de detección para el IDS Snort

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

Autor: Daniel Frias Mena

Tutor(es) : Ing. Yaima de los Ángeles Elías Álvarez
Ing. Yendry Machado García

La Habana, Cuba 2013

DECLARACIÓN JURADA DE AUTORÍA

Declaro que soy autor del resultado que expongo en la presente memoria titulada **Módulo de gestión de reglas de detección para el IDS Snort**, para optar por el título de Ingeniero en Ciencias Informáticas.

El presente trabajo fue desarrollado en el curso 2012-2013.

Finalmente declaro que todo lo expuesto se ajusta a la verdad, y asumo la responsabilidad moral y jurídica que se derive de este juramento profesional. Autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales del trabajo con carácter exclusivo.

Y para que así conste, firmamos la presente declaración jurada de autoría en La Habana a los 26 días del mes de junio del año 2013.

Autor: _____

Daniel Frias Mena

Tutor: _____ Tutor: _____

Yaima de los Ángeles Elías Álvarez Yendry Machado García

Resumen

La seguridad informática se ha convertido en un elemento fundamental en lo referente a la administración de redes de computación y los sistemas de detección de intrusiones (IDS por sus siglas en inglés), asumen un papel protagónico en el aseguramiento de entornos de redes. Snort es uno de estos sistemas de amplio uso hoy día. Su análisis se centra en el chequeo del tráfico de red contra las reglas de detección definidas. No existe una herramienta viable para la administración de estas reglas a pesar de ser un elemento crítico para el funcionamiento del IDS, constituyendo un proceso engorroso, manual y repetitivo.

La presente investigación propone una herramienta multiplataforma de gestión de reglas de detección de Snort, a la que incorporando un editor con un analizador léxico y sintáctico, sería capaz de brindar cierta asistencia al usuario en el proceso de creación y modificación de dichas reglas. Además, la propuesta de solución pretende eliminar la actual contrariedad del tratamiento manual de los ficheros de reglas, ofreciendo una interfaz que posibilite realizar operaciones básicas de gestión sobre los ficheros y sus reglas.

Palabras clave: IDS, reglas de detección de intrusiones, seguridad informática, Snort.

Tabla de Contenidos

Tabla de Ilustraciones

Índice de Tablas

Introducción

El surgimiento de la sociedad de la información y el incremento del uso de las tecnologías de información y las comunicaciones (TIC), impulsado por el desarrollo y crecimiento de la red de redes[1], hace que la información y los recursos informáticos que la gestionan tengan un rol principal en las actividades económicas, sociales y culturales. En el transcurso de los últimos 20 años se han integrado e informatizado, cada vez más, disímiles ámbitos de la vida cotidiana, entornos empresariales y gubernamentales. En estos contextos la información se convierte en un bien crítico que hay que proteger debido a la relevancia que puede llegar a alcanzar.

En contraste con lo anterior, los ataques que se producen a las aplicaciones y recursos informáticos se han incrementado notablemente. Las irrupciones exitosas que se realizan a las redes de compañías de alto perfil van en aumento (América-Economía, 2012). En nuestros días, personas con conocimientos medios en materia de informática y comunicaciones pueden ser capaces de realizar intrusiones en redes corporativas o estatales causando no solo pequeñas pérdidas, sino daños de millones de dólares en recursos y tiempo (Ponemon Institute, 2011). De la misma manera, es común la existencia de servidores que diariamente reciben algún tipo de agresión con diversos fines, desde detener sus servicios hasta obtener algún dato de tipo confidencial.

Consecuentemente, la seguridad en redes de computadoras ha pasado de ser una simple preocupación de unos pocos a un tema a analizar en casi cualquier entorno; la misma se ha vuelto imprescindible como forma de garantizar la integridad, disponibilidad y confidencialidad de la información. En tal sentido, existen numerosos sistemas dedicados a la seguridad que se destacan por su popularidad y utilidad, tales como: antivirus, *proxys*, cortafuegos (*firewall*), monitores de red y combinaciones de estos u otros tipos de sistemas de seguridad.

Una solución práctica para contrarrestar ataques y amenazas es establecer un cortafuego. Este cierra puertos y servicios determinados estableciendo cuál contenido ingresa en su dominio y cuál sale de este, reduciendo la posibilidad de una irrupción en la red; sin embargo, esta solución no es suficiente. El cortafuego representa una puerta que prohíbe el paso por puertos o protocolos a todos aquellos servicios no autorizados, pero permite el

[1] Debido a que Internet es un conjunto de redes interconectadas entre sí, este se conoce además como “Red de Redes”.

paso a aquellos servicios que necesitan ser utilizados por usuarios detrás de este. Es precisamente en la situación descrita que radica su debilidad.

En relación a lo descrito en el párrafo anterior, aunque un cortafuego solo permita el paso a los servicios básicos y teóricamente seguros, estos pueden presentar agujeros que se podrían aprovechar por un intruso; en este caso el cortafuego sería inútil ya que dejaría pasar algo que está "legalmente permitido", suponiendo que la integridad del cortafuego no haya sido comprometida antes. Por otro lado, se conoce que es posible la ocurrencia de ataques desde el interior de la red, en los que el cortafuego no interviene.

Del mismo modo que una puerta no es suficiente para proteger una casa y se instala un sistema de alarma para detectar cuando un ladrón ha conseguido acceso a la misma, un cortafuego no es suficiente para proteger una red. Por tanto, es necesario tener algún sistema que permita detectar una situación de este tipo, un vigilante, un IDS (Chirino Horta, y otros, 2007), (Bace, 2000). El propósito de estos sistemas, como su nombre lo indica, es la detección de intrusiones, que se define como el proceso de monitorizar redes de ordenadores y sistemas en busca de violaciones a las políticas de seguridad.

Los IDS aportan a un entorno interconectado la capacidad de prevención y de alerta ante cualquier actividad sospechosa. No están diseñados para detener un ataque pero pueden generar algún tipo de respuesta ante estos. Adicionalmente vigilan el tráfico de nuestra red, examinan los paquetes de red analizándolos en busca de datos sospechosos y son capaces de detectar las distintas fases de un ataque.

Existen numerosas soluciones comerciales y libres de IDS en el mercado actual. Snort es un sistema de detección y prevención de intrusiones, liberado bajo la licencia GPL[2]. Está basado en entornos de red, aunque con varios modos de ejecución, siendo capaz de realizar análisis de tráfico de red en tiempo real, combinando a su vez los beneficios de la inspección basada en firmas (como se explicará en el epígrafe 1.1.6), protocolos y anomalías. Adicionalmente, Snort es multiplataforma y cuenta con una amplia documentación; es una aplicación con cerca de 400 000 usuarios registrados actualmente (Sourcefire, 2013). De las características se deriva que dichos elementos le aporten a Snort gran flexibilidad y potencia de análisis. Como software, Snort presenta un desarrollo y

[2] Licencia Pública General de GNU o por su nombre en inglés General Public Licensees es la licencia más usada en el mundo del software y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el objeto.

crecimiento acelerado gracias a su comunidad y las funcionalidades que incorpora van en aumento.

Generalmente los IDS se construyen integrando componentes con funciones determinadas que interactúan entre sí para realizar el trabajo global del mismo (González Gómez, 2003). La eficacia de Snort depende en cierta medida de la gestión de reglas y la configuración de estas, de la manera en que estén escritas y hayan sido configuradas, pues inciden directamente en el funcionamiento del motor de detección. El incremento de la cantidad de reglas es directamente proporcional a la capacidad de procesamiento necesario para que el sistema responda en un tiempo aceptable. Conjuntamente, mientras la cantidad de reglas definidas sea alta, más probabilidades hay de que el sistema genere falsas alarmas frecuentemente (Rehman, 2003). De aquí que las reglas de detección se consideren un elemento esencial en el funcionamiento del sistema de detección de intrusos, al cual se le debe otorgar un tratamiento especial.

Al poner en funcionamiento a Snort es recomendable el empleo de herramientas de administración que faciliten la gestión de sus componentes y organicen la información generada por el sistema de detección. Estas herramientas presentan tablas que estructuran el contenido, generan diferentes tipos de gráficos, entre otras funciones específicas en dependencia del fabricante y la versión de dicha herramienta. De esta manera, hacen del manejo del IDS una tarea un poco más sencilla, incorporando interfaces gráficas de usuario (GUI por sus siglas en inglés) en sustitución de las complejas y poco intuitivas consolas. La mayoría de estas aplicaciones están liberadas bajo licencias de libre distribución.

Un elemento común en todas las herramientas examinadas, es que no brindan suficiente asistencia en la administración de reglas en cuanto a usabilidad y factibilidad. Generalmente, construir, desactivar y activar una regla implica trabajar directamente con los ficheros donde estas se almacenan. Al escribir o modificar una regla contenida en un fichero, el usuario solo se vale de los conocimientos y experiencia que tenga sobre reglas de detección; no cuenta con apoyo en dicho proceso, dígase completamiento de texto o resaltado de sintaxis incorrecta (codificación inteligente), aun cuando las reglas son un elemento crítico y de gran importancia para el funcionamiento del IDS. Por otra parte la cantidad de reglas existentes por defecto es elevada y tienden a aumentar con el tiempo. Por tanto, la administración de las reglas de detección es un proceso engorroso, manual y repetitivo, constituyendo una actividad en la que el usuario tiene muy poco apoyo, o ninguno.

Por lo anteriormente expuesto, se identifica como **problema de investigación**: ¿Cómo mejorar el proceso de gestión de las reglas de detección del IDS Snort? Para enmarcar los límites de esta investigación se especifica como **objeto de estudio:** los sistemas de detección de intrusos. Se define como **objetivo general**: Desarrollar un sistema que mejore el proceso de gestión de reglas de detección utilizadas por Snort a través de una interfaz gráfica que posea un editor con tecnología de codificación inteligente[3]. Se define como **campo de acción:** el proceso de gestión de reglas de detección del IDS Snort.

El objetivo general se desglosa en los **objetivos específicos** siguientes:

- Sintetizar las tendencias actuales referentes al proceso de gestión de reglas de detección para Snort.
- Determinar el diseño del sistema para su posterior implementación.
- Desarrollar una aplicación multiplataforma que permita la gestión de las reglas de detección del IDS Snort.
- Validar la implementación del producto.

Durante la investigación se hizo uso de los **métodos científicos** siguientes:

Analítico-Sintético: mediante este método se dividieron las características de las reglas del IDS Snort y los intérpretes de código en sus múltiples relaciones y componentes para facilitar su estudio. Igualmente, posibilitó establecer la unión de manera correlacionada entre las partes previamente mencionadas, descubrir sus propiedades generales y las relaciones esenciales entre ellas.

Modelación: este método permitió crear abstracciones de las interacciones entre los elementos del sistema y las estructuras de la gramática de las reglas de detección; lo que permitió una mayor comprensión entre dichos elementos debido a la correspondencia modelo-objeto presente en la investigación.

Justificación de la investigación

La actual investigación surge principalmente a partir de la necesidad de mejorar, en función de la usabilidad, el proceso de gestión de las reglas de detección de Snort. La misma se sustenta sobre la base de eliminar la actual ineficiencia de la gestión manual de los ficheros de reglas de detección. Al no contar con una asistencia en el proceso de creación y

[3] Tecnología que brinda asistencia en el proceso de escritura en un ordenador. Es muy aplicada en las aplicaciones IDE (ver epígrafe 1.5.4) posibilitando el resaltado de sintaxis del lenguaje, auto-identación, completamiento de código, resaltado de errores en tiempo real entre otros.

modificación de las reglas, se espera facilitar su correcta definición, brindando tecnología de codificación inteligente por medio de un editor. Lo anterior tributa a la posibilidad de disminuir los errores, con lo cual no se comprometería la eficacia del motor de detección; además de que se crea un ambiente seguro para el manejo de los ficheros de reglas. Cabe mencionar que Snort es ampliamente usado por administradores de redes de todo el mundo y es la solución IDS que se utiliza en la Universidad de las Ciencias Informáticas (UCI[4]), lo cual permitiría probar la solución desarrollada en diferentes entornos reales y le atribuye a la presente investigación mayor utilidad práctica y social.

El documento consta de los capítulos siguientes:

Capítulo 1 Basamentos Teóricos de la Detección de Intrusiones y Gestión de Reglas de Detección: Se establece el marco conceptual que crea las bases relacionadas al objeto de estudio de la investigación. Debido a la existencia actual de múltiples herramientas, se realiza un análisis del estado del arte resaltando los resultados del mismo. Se lleva a cabo además, la fundamentación de las herramientas y tecnologías empleadas durante el proceso de desarrollo de la aplicación.

Capítulo 2 Planificación y Diseño del Módulo de Gestión de Reglas de Detección: Se realiza un análisis detallado del problema a resolver. Luego, son definidos los requerimientos del cliente a través de las historias de usuario y la planificación del ciclo de desarrollo de la aplicación; dando paso al diseño de la propuesta de solución.

Capítulo 3 Construcción y Pruebas del Sistema: Se describen los componentes fundamentales en la implementación de la aplicación. Describiéndose elementos como la definición de los estándares que guían la codificación y una descripción de la interfaz de usuario de la aplicación. Igualmente se explica la validación del sistema describiendo el proceso de pruebas llevado a cabo durante toda la producción de la aplicación.

[4] A partir de este punto el autor establece que Universidad se refiere a la UCI

Capítulo 1 Basamentos Teóricos de la Gestión de Reglas de Detección con Snort

Para el desarrollo del sistema es necesario definir, primeramente, una serie de conceptos básicos en aras de crear un soporte teórico que ayude al entendimiento del domino de la investigación. Así como, realizar un estudio de las aplicaciones actuales de gestión para Snort, en aras de obtener una panorámica del estado del arte respecto a la gestión de reglas. Adicionalmente es esencial la identificación de las tecnologías de desarrollo y herramientas a emplear en la construcción de la aplicación.

1.1 PRINCIPALES CONCEPTOS REFERENTES A LA GESTIÓN DE REGLAS

Con el objetivo de lograr un mejor entendimiento de todos los elementos asociados a la presente investigación, en el presente epígrafe se relacionan y se definen los conceptos fundamentales que se utilizan durante el desarrollo del trabajo.

1.1.1 SEGURIDAD

En orden con la fuente (Real Academia Española, 2012) se tiene que, con el fin de garantizar que todo aquello que represente un elemento de vital importancia esté debidamente protegido, se define la palabra seguridad. Esta proviene del latín *securitas*, que a su vez se deriva de *securus*, significa sin cuidado, sin preocupación, libre de cualquier peligro o daño. Igualmente denota mecanismo que asegura el buen funcionamiento de algún elemento, precaviendo que este falle, se frustre o se violente.

La seguridad se redefine en muchos ámbitos de la vida y una de estas aristas es la información. En correspondencia a los autores (Contrera Chávez, 2011) y (Fiebel, 2005) se tiene que la seguridad de la información es un concepto general, y se entiende por todas aquellas medidas preventivas y reactivas para proteger la información buscando mantener la confidencialidad, disponibilidad e integridad de la información. No obstante, es común confundir el concepto anterior con la seguridad informática, aunque es irrefutable su estrecha relación. Específicamente la seguridad informática se manifiesta sobre la información en sistemas informáticos y componentes de los mismos, encargándose de asegurar y proteger la infraestructura y los datos de dichos sistemas; manteniendo los principios establecidos por la seguridad de la información (Ramió Aguirre, 2004). Un sistema informático se puede proteger en dos enfoques: físico (relacionado con la protección del *hardware*), y lógico (referente al aseguramiento del *software* y la información

que manejan). La seguridad informática tiene muchas especialidades y aristas: una de ellas son los sistemas de detección de intrusiones.

1.1.2 INTRUSO

Un intruso en una red puede ser llamado popularmente *hacker* o *craker*, pero esta afirmación no es del todo correcta debido a que no existe un consenso único en cuanto a qué se refieren estos términos. Explícitamente, un intruso en una red informática es aquella persona o programa informático que burla los principios de la seguridad informática. Los intrusos utilizan las deficiencias o agujeros de seguridad en un sistema para violar las políticas de seguridad establecidas. Estas incorrecciones son conocidas como vulnerabilidades. Cuando un intruso aprovecha una vulnerabilidad se crea una situación en la cual existe la posibilidad potencial de perjudicar o dañar al sistema informático. Esto es conocido como amenaza. Se pueden consultar las fuentes (Allan, 2005) y (esCERT, 2009) en complementación a las definiciones dadas en este epígrafe.

1.1.3 SISTEMA DE DETECCIÓN DE INTRUSOS

La Detección de Intrusiones es el fruto de la aplicación del Procesamiento Electrónico de Datos[5] a las auditorías de seguridad, utilizando mecanismos de identificación de patrones y métodos estadísticos. Igualmente establecen un conjunto de técnicas y métodos que son empleados en la detección de actividades sospechosas a nivel de red o de ordenador. Según (González Gómez, 2003) Luego, un sistema de detección de intrusos es un *software*, *hardware* o combinación de ambos usados en la detección o identificación de cualquier actividad dudosa.

Los sistemas de detección de intrusiones están compuestos por tres elementos fundamentales y Snort por supuesto no está exento de esto:

- Una fuente de información que proporciona eventos del sistema.
- Un motor de análisis que busca evidencias de intrusiones.
- Un mecanismo que actúa según los resultados del motor de análisis.

Estos elementos conforman la arquitectura principal de todo IDS. Entre ellos fluye la información y cada parte interactúa con otra u otras hasta llegar a los mecanismos de reporte, respuestas autónomas o de notificación. Un sistema de detección de intrusos

[5] EDP por sus siglas en inglés, es el proceso de generar, almacenar y revisar eventos de un sistema informático cronológicamente.

puede clasificarse según cuatro categorías: fuente de información, tipo de análisis, estructura y respuesta; la Figura 1 muestra la clasificación que puede recibir el sistema atendiendo a cada una de estas categorías.

Los IDS de red (*Network IDS* o *NIDS* por sus siglas en inglés) capturan los paquetes de datos que circulan por determinada red en busca de elementos que indiquen un ataque. Pueden situarse en equipos de interconexión u ordenadores para analizar todo el tráfico que pase por el segmento de red donde se ubiquen. Normalmente utilizan dispositivos de red en modo promiscuo[6], convirtiendo al sistema en un rastreador (*sniffer*) (Allan, 2005). En el caso de los sistemas basados en máquina (*Host IDS* o *HIDS* por sus siglas en inglés) recogen los datos generados por un ordenador, normalmente a nivel de sistema operativo. Los IDS de aplicación registran la actividad de un determinado software, por ejemplo los registros de un servidor FTP[7]. Aquellos sistemas de detección que combinan dos o más fuentes se categorizan como híbridos.

De acuerdo al tipo de respuesta que un IDS puede emitir o ejecutar, se clasifica como activo cuando genera algún tipo de acción sobre el sistema atacante o fuente de ataque como cerrar la conexión o enviar algún tipo de réplica predefinida en la configuración. Son pasivos los que se limitan en notificar a la autoridad competente o administrador de la red mediante una alerta, mensaje, archivo de *log*, entre otros tipos definidos (Bace, 2000). Pero no actúa sobre el ataque o atacante.

Según el tipo de estructura los IDS se clasifican en distribuidos y centralizados. Un sistema centralizado es aquel donde las tareas de monitoreo y control son realizadas desde un único lugar. La ventaja de un mecanismo centralizado es la facilidad para desarrollarlo y mantenerlo, pero significa un cuello de botella cuando el tráfico en la red es grande y además se convierte en un único punto de falla. En un sistema distribuido el mecanismo de detección y análisis se puede realizar por diversos agentes cooperantes y autónomos a los cuales puede asignárseles tareas específicas de detección (Britos, 2010). La principal ventaja de los sistemas distribuidos es que los agentes trabajan en paralelo y pueden reaccionar en forma rápida al tráfico cambiante de la red.

Snort es un IDS basado en red logrando funcionar en tres modos principales: como rastreador permite percibir en tiempo real qué ocurre en la red y todo el tráfico; en modo

[6] Es aquel en el que un equipo de interconexión o un ordenador conectado a una red captura todo el tráfico que circula por ella.
[7] File Transfer Protocol, FTP por sus siglas en inglés, Protocolo de Transferencia de Archivos.

registro de paquetes posibilita guardar en un archivo el registro de los mismos para su posterior análisis; y como un NIDS propiamente, constituyendo la configuración más compleja pero completa, permitiendo el análisis de todo el tráfico de red ejecutando acciones definidas basadas en la observación que realiza (Roesch, y otros, 2012). Además, según su estructura constituye un IDS distribuido debido a que puede valerse de varias instancias corriendo en dispositivos con ubicaciones distintas dentro de la red, para recopilar información.

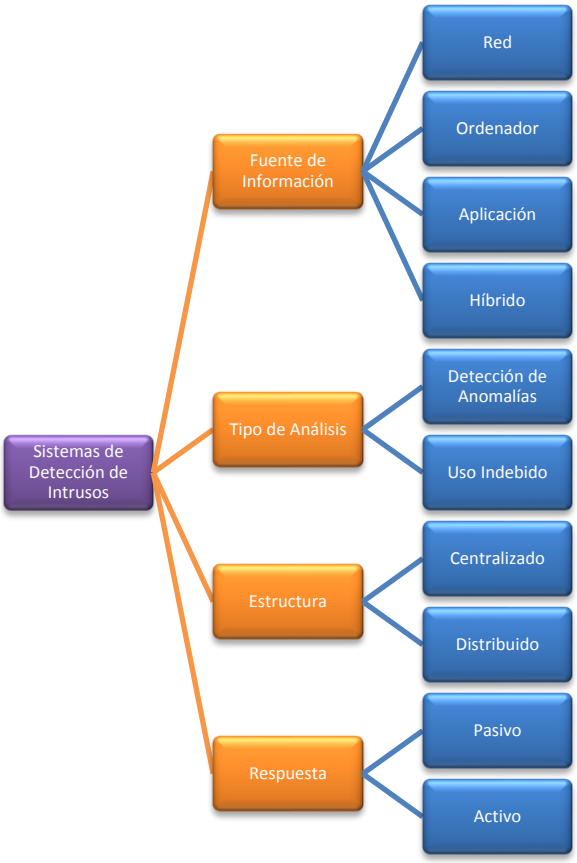


Figura 1 Clasificación de los sistemas de detección de intrusos.

1.1.4 ALERTAS

Las alertas son todo tipo de notificaciones de una actividad intrusiva detectada. Cuando un IDS detecta un intruso, este informa al administrador de seguridad del incidente usando alertas. Estas pueden ser en forma de ventana de globo (*pop-up*), un registro en un archivo, un e-mail o cualquier tipo de aviso. Pueden ser almacenadas en archivos o en bases de datos, las cuales pueden ser consultadas por responsables de seguridad (García Alfaro, 2005).

Snort puede generar alertas en muchos formatos y enviarlas a distintos destinos. Algunos módulos que contiene pueden incluso modificar la configuración de un cortafuego para bloquear un puerto, conexión u ordenador, todo en relación a la alerta emitida por el IDS.

1.1.5 FALSOS POSITIVOS Y FALSOS NEGATIVOS

Las determinaciones referente a los falsos positivos y falsos negativos que se expresan a continuación están en concordancia con las fuentes (Tori, 2008), (Scott, y otros, 2004) y (Lucena López, 2005). Los falsos positivos son alertas generadas por un sistema de seguridad, por ejemplo un IDS; pero dicho sistema lo entiende como un ataque verdadero contra el dominio que protege, por ejemplo una red; sin embargo el ataque no se está llevando a cabo realmente. La ocurrencia de estos tipos de alertas no es conveniente, pues pueden crear tráfico innecesario en la red, y así ocultar, en términos de sistemas de detección de intrusiones, el ataque real. De la misma manera, hacen perder tiempo y recursos a los responsables de seguridad, al buscar y tratar de corregir irrupciones que nunca sucedieron.

Los falsos negativos son ataques reales que no son detectados por el sistema de seguridad, y en el caso de Snort, no generaría ninguna alerta. Existe la posibilidad que un IDS no reconozca un ataque o irrupción, por ejemplo, porque se encuentra sobrecargado de contenido de procesamiento a causa de que el atacante ha usado satisfactoriamente algún método para evadir al sistema de detección o debido a una mala configuración de sus componentes de análisis como las reglas de detección. La implicación de esta situación es obvia: un ataque que el IDS pasa por alto, es un ataque que llega a su objetivo.

1.1.6 TIPO DE ANÁLISIS

Existen dos categorías básicas de técnicas de detección de intrusiones: detección de usos indebidos y detección basada en anomalías. La detección de usos indebidos (*misuse*) se basa esencialmente en la definición de elementos incorrectos en el contenido de los paquetes de red. Esta definición contiene descripciones de ataques o acciones dudosas conocidas como firmas, por ejemplo, combinaciones dentro de los paquetes de red que no sean legítimas. Estos elementos sospechosos contienen firmas como las contenidas en los virus informáticos, que son detectadas por los antivirus (Rehman, 2003). Basado en el conjunto de estas signaturas y reglas de detección establecidas, el sistema de detección es

capaz de comprobarlas contra el flujo de datos auditados, buscando y registrando evidencias de ataques conocidos. La Figura 2 esquematiza lo planteado anteriormente.

Figura 2 Detección basada en usos indebidos (González Gómez, 2003).

La detección basada en anomalías se basa en modelos de comportamiento previamente establecidos. La desviación del comportamiento "normal", se interpreta como una posible irrupción en el sistema o actividad "anormal". En este tipo de análisis se utilizan diversos algoritmos para crear perfiles de comportamiento normal, midiendo la actividad de los actores sobre los objetos de un entorno. Dicho perfil se actualiza con información nueva dinámicamente. Se construyen modelos a contrastar con el comportamiento actual de un usuario, ordenador, o conexión de red (González Gómez, 2003). Las desviaciones que resultan de esta comparación son sometidas a técnicas que utiliza el sistema para decidir si ha ocurrido o no indicios de intrusiones. Estos sistemas emplean procesos estadísticos y

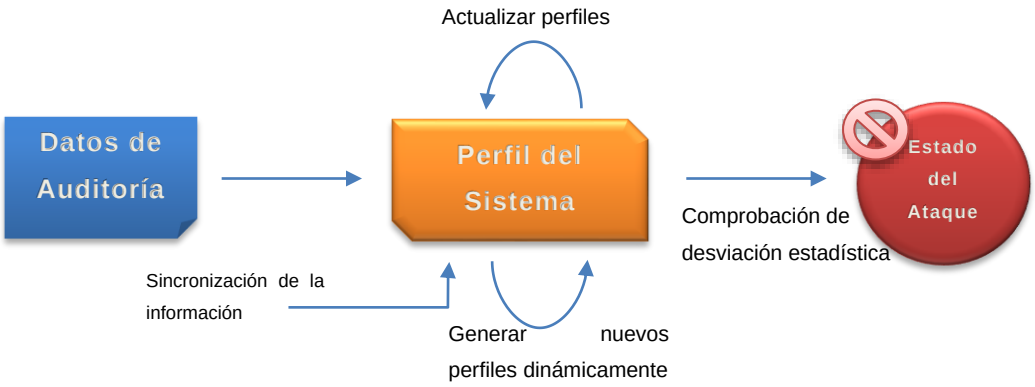


Figura 3 Detección basada en anomalías (González Gómez, 2003).

técnicas de inteligencia artificial, como redes neuronales y agentes inteligentes autónomos (autómatas). El proceso es ilustrado en la Figura 3.

Snort realiza su análisis de paquetes mediante la detección de usos indebidos principalmente, dado que es un sistema basado en reglas y conjuntamente comprueba firmas en los paquetes analizados. Sin embargo posee módulos (*plug-ins*) que son capaces de detectar anomalías en las cabeceras de los protocolos de red.

1.2 COMPONENTES DE SNORT Y SU FUNCIONAMIENTO

En el proceso de diseño de Snort, los desarrolladores centraron sus esfuerzos en tomar una herramienta existente y ampliar sus capacidades en gran medida para hacer algo diferente y aprovechable. Lo señalado por el autor (Scott, y otros, 2004), define que la herramienta existente era *tcpdump*, el captor de paquetes presente en muchos sistemas Unix. Marty Roech creador de Snort, tomó de *tcpdump* la capacidad de capturar los

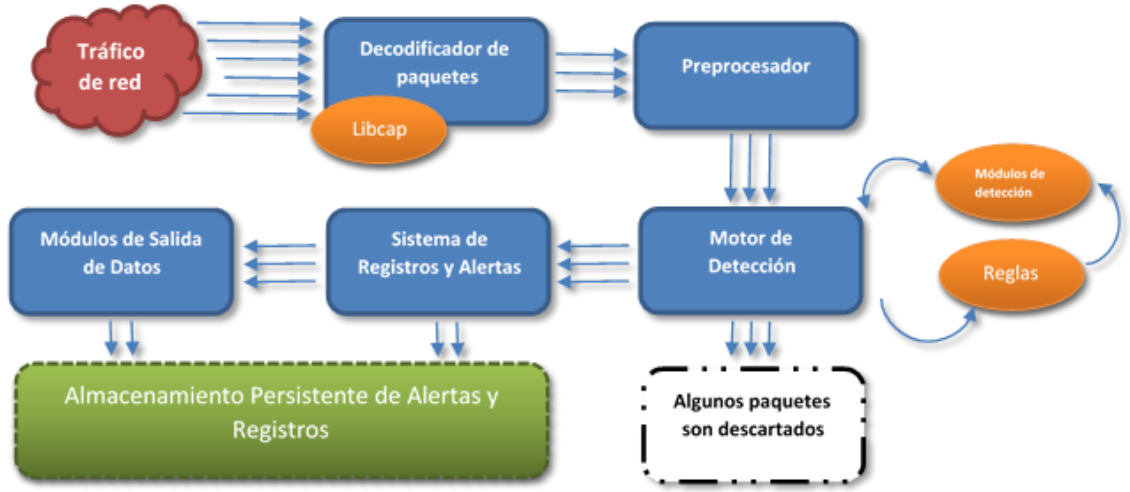


Figura 4 Interacción de los componentes principales de Snort

paquetes de la red, y añadió la función de examinarlos contra un conjunto de firmas. El desarrollo progresivo del sistema permitió conformar los componentes principales de Snort, estos le permiten realizar su actual funcionamiento. Cada componente desarrolla una tarea específica trabajando de manera correlacionada entre ellos para darle tratamiento a la información examinada. Lo anterior se muestra en la Figura 4.

Los paquetes del tráfico de red llegan al preprocesador a través la librería *libcap*, la cual captura los mismos directamente del tráfico. El decodificador está compuesto por varios decodificadores, de modo que cada uno descifra elementos de protocolos específicos provenientes de los paquetes capturados, separando los paquetes en partes pequeñas (Pino, 2009). Primero se decodifica la información perteneciente a la capa de enlace[8].

[8] Segunda capa del modelo OSI, que especifica cómo se organizan los datos cuando se transmiten en un medio particular como *Ethernet*, *TokenRing* u otros.

Luego decodifica la información perteneciente al protocolo IP y por último las partes referentes a TCP o UDP. Cuando la decodificación finaliza, Snort tiene la información de todos los protocolos del paquete analizado, almacenada en una estructura de datos, para que sea tratada por el preprocesador.

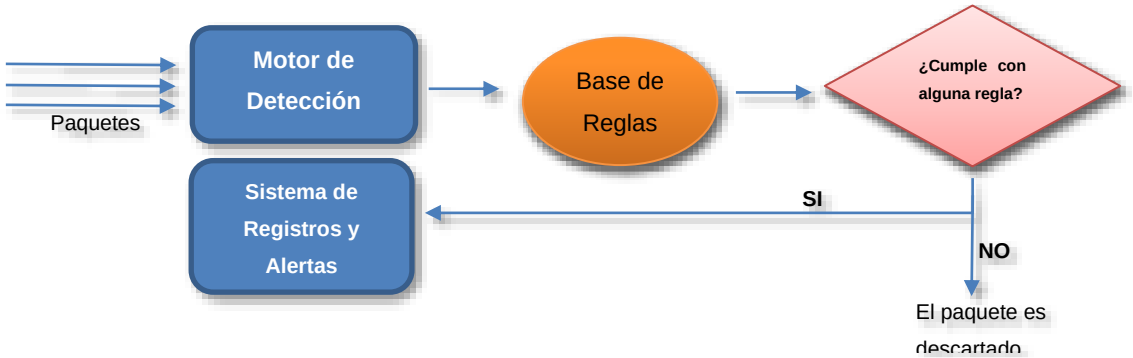


Figura 5 Funcionamiento básico del motor de detección de Snort (García Alfaro, 2005).

El preprocesador está conformado por varios componentes o módulos utilizados por Snort para organizar o modificar el orden de los paquetes antes que el motor de detección realice operaciones sobre estos. Por ende su propósito central es eliminarle trabajo excesivo al motor. Estos módulos pueden ser activados y desactivados según la necesidad del nivel de pre procesamiento. En concordancia con (Rehman, 2003), si el número de preprocesadores activados es muy alto, el rendimiento de Snort puede caer notablemente, debido a que cada módulo ejecutará un análisis o nidificación sobre el contenido procesado.

El núcleo central de Snort y su componente fundamental es el motor de detección. Su tarea es detectar cualquier actividad intrusiva a partir de la información proporcionada por el preprocesador y los módulos asociados. Para ello el motor de detección contrastará los datos con su base de reglas, si alguna coincide con la información obtenida, el motor de detección notifica al sistema de registros y alertas del evento. En caso de que un paquete no coincida con ninguna regla, este es descartado y no se toma en cuenta en ningún proceso posterior. En la Figura 5 se resume el comportamiento general del motor de detección.

Como resultado del procesamiento de los paquetes por los componentes previos al motor de detección, este puede diseccionar un paquete y aplicar reglas a diferentes partes del mismo, como son la cabecera del protocolo IP, la cabecera de la capa de transporte (TCP, UDP, ICMP), el contenido de la capa de aplicación (DNS, FTP, otros) y el cuerpo o contenido del paquete (*payload*).

De acuerdo a los planteamientos de (Pino, 2009) y (Roesch, y otros, 2012) el motor de detección es el componente crítico de Snort, en cuanto a tiempo. Dependiendo de la potencia de procesamiento del hardware y de cuántas reglas estén definidas, su trabajo tomará un tiempo variable. Cabe mencionar que los conjuntos de reglas pueden ser activadas y desactivadas para definir el comportamiento de detección y liberar carga de procesamiento según el tipo de red donde Snort esté configurado. La carga de trabajo del motor de detección depende de los siguientes factores:

- Número de reglas de detección activas.
- Capacidad de procesamiento del ordenador donde Snort se ejecuta.
- Velocidad del bus interno usado en el ordenador donde Snort se ejecuta.
- Carga de la Red.

Los resultados del análisis del motor de detección se registran mediante el sistema de registros y alertas, el cual puede generar resultados en distintos formatos y hacia distintos equipos. Cuando una alerta es lanzada por el motor de detección, se genera una acción que puede suponer la generación de un fichero de registro (*log*), o que esta sea enviada mediante un mensaje SNMP , o ser almacenada de forma estructurada por algún sistema gestor de bases de datos.

Es posible además la utilización de herramientas alternativas desarrolladas por terceras partes, que facilitan la visualización y el tratamiento de la información reportada por Snort, las que pueden ser incorporadas a los módulos de salida de datos. Estas pueden realizar diferentes operaciones, dependiendo de cómo se desee guardar la salida generada por el sistema de alertas y registros. En dependencia de la configuración, la información que estos módulos pueden registrar es variada: cuándo el ataque fue detectado, tipo de ataque, de dónde viene, a dónde va, dirección MAC[9], dirección IP, contenido del paquete entre otros. Además pueden ejecutar funciones como crear registros simples en el disco duro, enviar mensajes SNMP, registrar informes en bases de datos, generar informes XML, modificar configuraciones en un *router* y cortafuego entre otros (Rehman, 2003).

[9] Frecuentemente llamada dirección física. Media Access Control por sus siglas en inglés su traducción es Control de Acceso Medio. Es un identificador de 48 bits (6 bloques hexadecimales) que corresponde de forma única a una tarjeta o dispositivo de red.

1.3 REGLAS DE DETECCIÓN

Las reglas son una metodología diferente para realizar detecciones, que brindan la ventaja de constituir una protección contra ataques de día-0[10]. A diferencia de las firmas, las reglas se basan en detectar principalmente el ataque, no una amenaza o un dato específico (Sourcefire Inc, 2010). De modo práctico, una regla de detección consiste en un conjunto de requisitos que permite en caso de cumplirse, realizar una acción determinada. Para conformar una regla se requiere un agudo conocimiento respecto a las características de una vulnerabilidad determinada y cómo se manifiesta en caso de materializarse.

Cada una de estas reglas está asociada a un identificador único que permite reconocer y encontrar información acerca del ataque o mal uso detectado. Además, gracias al uso mayoritario de Snort entre la comunidad de administradores de red, se ha estandarizado el formato de construcción de reglas, aumentando a diario el soporte de las mismas (García Alfaro, 2005). De este modo, cualquier usuario de Snort, o de cualquier otro IDS con un formato de reglas compatible, podría crear sus propias reglas a medida que vayan apareciendo nuevos ataques colaborando para mantener actualizada su base de firmas.

A modo de ejemplo, se puede escribir una regla que permita verificar el uso de aplicaciones punto a punto[11] (*peer-to-peer*) para el intercambio de ficheros a través de Internet, verificando el uso de la cadena GET[12] en solicitudes diferentes al puerto 80 tradicional del protocolo HTTP. Si un paquete capturado coincide con esta especificación, el sistema de notificación lanzará una alerta.

Las reglas son un elemento fundamental en la eficacia del motor de detección. De su correcta escritura depende el número de falsos positivos generados por el sistema y estos no son beneficiosos para la seguridad. Para evitarlos, en ocasiones se necesita modificar o desactivar un conjunto de estas reglas. De la misma manera, mientras la cantidad de reglas definidas aumenta, se necesita más capacidad de procesamiento para capturar los datos en tiempo real. Por ello es importante capturar la mayor cantidad de anomalías con la menor cantidad de reglas (Rehman, 2003).

[10] Un ataque de día-0 se realiza contra una aplicación o sistema el cual debe el éxito a la existencia de vulnerabilidades que son desconocidas para los responsables de seguridad, fabricantes del producto y usuarios en general. Esto supone que dichas vulnerabilidades aún no hayan sido reparadas en el momento del ataque.
[11] Red de computadoras donde los elementos definidos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red.
[12] Comando del protocolo HTTP que solicita el recurso ubicado en la URL especificada.

1.3.1 ESTRUCTURA

Una regla consta de dos partes: la cabecera y el cuerpo. Snort establece el control de los archivos de reglas y categoría de estas a través del archivo *snort.conf.* Este archivo tiene dos entradas importantes respecto a la configuración de las reglas (Scott, y otros, 2004):

- La ubicación del directorio de reglas, leída del archivo en el momento de inicialización de Snort. En esta entrada la variable *$RULE_PATH* debe contener la dirección de la ubicación de las reglas. Por ejemplo: `/usr/local/snort/rules` en Linux o `C:\Program Files\snort\rules` en Windows.
- En la parte final del archivo se pueden ver todos los conjuntos de reglas. Se puede desactivar toda una categoría de reglas comentando la línea asociada a la misma, lo cual permite añadir o eliminar clases enteras de reglas. Por ejemplo:

```
Include $RULE_PATH/local.rules
Include $RULE_PATH/bad-traffic.rules
Include $RULE_PATH/exploit.rules
```

1.3.2 CABECERA DE LA REGLA DE DETECCIÓN

La cabecera es un filtro frontal que separa el tráfico en cinco factores diferentes: dirección IP origen, dirección IP destino, puerto origen, puerto destino y protocolo, ayudando a filtrar el tráfico de red. La cabecera se compone de los elementos que se muestran en la Tabla 1.

Acción	Protocolo	Origen	Puerto	Dirección	Destino	Puerto
log, alert, pass, activate, dynamic, drop, reject, sdrop	TCP, UDP, IP, ICMP	Dirección IP, CIDR[13], $EXTERNAL_NET, $HOME_NET	any, 3306, $HTTP_PORTS	-> <>	Dirección IP, CIDR, $EXTERNAL_NET, $HOME_NET	Any, 3306 $HTTP_PORTS

Tabla 1 Estructura de la cabecera de la regla de detección.

[13] Classless Inter-Domain Routing o CIDR (en español enrutamiento entre dominios sin clases) permitió una mayor flexibilidad al dividir rangos de direcciones IP en redes separadas. Permite un uso más eficiente de las cada vez más escasas direcciones IPv4 y un mayor uso de la jerarquía de direcciones, disminuyendo la sobrecarga de los enrutadores principales de Internet para realizar el encaminamiento.

La acción es un factor que permite indicar la operación que se debe realizar sobre el paquete que concuerde con la regla. Los valores admisibles son (Roesch, y otros, 2012):

- ***Log**:* añade el paquete malicioso al registro de salida establecido en la configuración del sensor[14]. Las opciones del módulo de salida son muy variadas posibilitando diferentes elecciones posibles. Las directivas de *log* por cada regla permite personalizar el registro del paquete a un nivel notable.
- ***Alert**:* esta acción puede imprimir una entrada de *log* y lanzar una notificación con algún evento asociado cuando una regla es activada. La acción de alerta es la establecida por defecto en el paquete de reglas de Snort.
- ***Pass**:* puede descartar un paquete que concuerde con la regla y continuar con el procesamiento. Es útil cuando se está poniendo a punto el paquete de reglas y se necesita que las más problemáticas sean descartadas, en aras de ver las salidas de las reglas de más interés, lo cual además permite reducir el número de falsos positivos.
- ***Activate**:* es la acción más potente de Snort. Esta realiza su operación de acuerdo a la acción *dynamic*, activando una alerta y accionando lo especificado por la regla *dynamic* asociada. El par *activate/dynamic* es ideal para capturar complejas series de ataques que de otra manera pudieran no ser contemplados.
- ***Dynamic**:* esta acción está asociada con una regla que no debería activarse hasta que un evento determinado ocurra. Se combina con la acción *activate* para establecer un segundo nivel de procesamiento en determinadas circunstancias. Esta combinación además puede ser una herramienta útil para una detección de intrusiones avanzada.
- ***Drop**:* bloquea el paquete y lo registra, enviando la información a los módulos de salida.
- ***Reject**:* esta acción bloquea el paquete, lo registra, luego reinicia la conexión si el protocolo es TCP, y si es UDP envía un mensaje de puerto ICMP inalcanzable.
- ***Sdrop**:* es bien parecida a *drop* pero no registra el paquete, se limita solo a bloquearlo con esta acción.

Snort como un IDS de red debe operar en los niveles más bajos de la red para realizar su trabajo, capturando los paquetes de la parte física de la red. En este marco, Snort realiza

[14] En el documento se refiere a sensor como un ordenador o equipo de cómputo donde se ejecute Snort. En cualquier otro coso se aclara explícitamente su significado.

un escaneo de los protocolos IP, ICMP, TCP y UDP (Roesch, y otros, 2012). Para construir una regla es necesario especificar uno de estos cuatro protocolos.

La parte del origen/destino de una regla de Snort es muy importante en la conformación de la misma. Puede ser una dirección IP común o empleando la notación CIDR. La sección origen/destino de una regla se especifica con una de las variantes siguientes:

- Red (10.35.24.0/24, red de clase C con 254 host posibles, sin contar con la IP 10.35.24.0 y 10.35.24.254 ya que están reservadas para identificador de red y difusión respectivamente)
- Host (se especifica como 10.35.24.102 ya que es una dirección IP)

Cabe mencionar que origen y destino, en sustitución de las dos variantes anteriores, pueden recibir las variables $EXTERNAL_NET y $HOME_NET que se definen en el archivo de configuración *snort.conf* de Snort. Estas variables se denominan argumentos y se establecen al inicio del archivo mencionado y comienzan con $. Lo anterior provee mayor legibilidad al manejar largas colecciones de reglas.

Para el puerto se define el número por el cual se establece la comunicación del 0 al 65535 (16 bits); por ejemplo el 80 para las conexione HTTP, el puerto 443 para las conexiones encriptadas HTTP o el 25 para el envío de e-mail. De la misma manera puede establecer un rango de puertos como 80:100 lo que significaría referirse a los puertos del 80 al 100; o :100 que serían los puertos del 0 al 100, al igual que 100: que serían del 100 al 65536. Los puertos además se pueden negar; por ejemplo si se quisiera establecer en la regla, que concuerde con todos los puertos excepto el 6645, se colocaría el carácter “**!**” delante y quedaría como !6645.

El operador de dirección indica la orientación o dirección del tráfico al que la regla se aplica. Estos pueden ser -> y <>. El primero significa que la dirección IP y puerto a la izquierda es el origen y los mismos elementos a la derecha es el destino de la comunicación. El segundo es un operador bidireccional, lo cual establece que la regla concuerde con cualquier tráfico de red entre lo señalado a la izquierda y derecha de este operador, independientemente de donde se haya originado. Es útil para analizar los dos lados de la conexión como una sesión

de telnet[15] o POP3. Se pueden consultar las fuentes (Pino, 2009), (Scott, y otros, 2004) y (Roesch, y otros, 2012) para ampliar la información brindada en este epígrafe.

1.3.3 CUERPO DE LA REGLA DE DETECCIÓN

El cuerpo de la regla está compuesto de opciones escritas dentro de dos paréntesis primarios separadas por punto y coma. El formato de la mayoría de las opciones es una palabra clave, luego dos puntos seguido del valor asignado a la palabra clave. El valor puede aparecer entre comillas, por ejemplo: `ítem:"valor"`. Al final de la última opción antes del paréntesis cerrado se coloca un punto y coma para delimitar que ese paréntesis se corresponde con el fin de la regla. Todas las palabras clave reconocidas que se pueden ubicar dentro del cuerpo de la regla se describen en el manual de usuario de Snort (Roesch, y otros, 2012).

Existen cuatro grandes categorías de las opciones de una regla:

- **General***:* Proveen información sobre la regla pero no tiene ningún efecto en la detección.
- **Payload***:* Buscan datos específicos dentro de la carga útil del paquete.
- **Non-Playload***:* Buscan datos específicos en los campos que no son carga útil, por ejemplo la cabecera.
- **Post-Detection***:* Permiten activar reglas específicas luego de que otra sea activada.

1.3.4 EJEMPLO DE DEFINICIÓN DE UNA REGLA DE DETECCIÓN

Para ilustrar la construcción de una regla de detección se tiene el siguiente ejemplo:

- ```
  alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS
  CodeRed v2 root.exe access"; flow:to_server,established;
  uricontent:"/root.exe"; nocase; classtype:web application-attack;
  reference:url,www.cert.org/advisories/CA-2001 19.html; sid:1256; rev:7;)
  ```

La directiva ***alert*** establece que si la regla es activada, se envía información a los componentes restantes en forma de alerta. La mayoría de las reglas de Snort emplean esta directiva. El elemento ***tcp*** establece el protocolo de red al que se aplica la regla, lo cual indica que la misma analizará el tráfico de red que emplea el protocolo TCP, ignorando el resto del tráfico. La dirección es determinada en la regla descrita como cualquier paquete

[15] Es el nombre de un protocolo de red a otra máquina para manejarla remotamente como si estuviéramos sentados delante de ella.
  ```

que se origina en *$EXTERNAL_NET* (argumento donde está definido la red externa), por cualquier puerto (*any*), cuyo destino son los servidores Web (utilizando el argumento *$HTTP_SERVERS*) por los puertos establecidos en el argumento *$HTTP_PORTS*.

Los elementos ubicados entre el par de paréntesis son las opciones de la regla, las cuales aportan información específica a coincidir con el tráfico analizado por la regla. La opción ***msg*** comunica al sistema de registro y alertas el mensaje a establecer como registro, como alerta o ambos. Es un texto simple que debe usar caracteres de escape con el símbolo \ si se quieren incorporar caracteres especiales para no causar errores (Roesch, y otros, 2012). La etiqueta ***flow*** define que la conexión debe ir al servidor y ya debe estar establecida de acuerdo a la propiedad *stablished*. La propiedad ***uricontent*** busca dentro del campo URI[16] de respuesta normalizado. El hecho que este normalizado indica que los caracteres recurrentes y especiales, serán removidos de la información. En el caso del ejemplo se busca que el campo contenga el texto `"/root.exe"`. Las opciones ***flow*** y ***uricontent*** ayudan en el refinado de las reglas de detección.

Los ataques se pueden clasificar definiendo un valor en la opción ***classtype*** facilitando la categorización de las reglas de detección en tipos de ataques generales y más específicos para obtener una mayor organización y entendimiento de los eventos generados. Estas clasificaciones de ataques residen en el archivo *classification.config* el cual se puede editar para añadir nuevas clasificaciones. El valor de la palabra clave ***sid*** establece un identificador único para una regla. Esta información le permite a los módulos de salida identificar la regla unívocamente. Note que este elemento debe ser usado conjuntamente con ***rev.*** La misma se utiliza en el establecimiento del número de revisión de una regla de Snort. Estas dos opciones permiten que la firmas y descripciones sean refinadas y se puedan reemplazar con información actualizada (Roesch, y otros, 2012).

La palabra clave ***reference*** define una entrada relativa a cualquier otra fuente con información referente al ataque identificado. Generalmente son URLs de sitios web temáticos referentes a la seguridad informática, los cuales contienen consejos con información al ataque o sobre cómo resolver el hueco de seguridad que posibilita su realización. La utilización del ***nocase*** establece que Snort no diferencie las mayúsculas en los textos que busca. La Tabla 2 muestra la estructura de la regla del ejemplo descrito.

[16] Uniform Resource Identifier o URI (en español identificador uniforme de recursos) es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

Tabla 2 Estructura del ejemplo de regla descrito (Scott, y otros, 2004).

Cabecera					Cuerpo
Acción	Protocolo	Dirección fuente y puerto	Flujo	Dirección destino y puerto	Opciones
alert	tcp	$EXTERNAL_NET any	->	$HTTP_SERVERS $HTTP_PORTS	(msg:"WEB-IIS CodeRed v2 root.exe access"; flow:to_server,established; uricontent:"/root.exe"; nocase; classtype:web application-attack; reference:url,www.cert.org/advisories/CA-2001 19.html; sid:1256; rev:7;)

1.4 ESTADO DEL ARTE DE LA GESTIÓN DE REGLAS EN LAS SOLUCIONES DE ADMINISTRACIÓN EXISTENTES PARA SNORT

Con el fin de examinar las soluciones y tendencias actuales referentes a la gestión de reglas de detección para Snort, se realizó un estudio de varias herramientas existentes para este fin. Esto permitió conocer el estado actual respecto a la gestión de reglas de detección de Snort, así como aspectos positivos y deficiencias a tener en cuenta en el proceso de desarrollo del sistema que se propone.

1.4.1 IDS POLICY MANAGER 3.0

IDS Policy Manager 3.0 (IDSPM por sus siglas en inglés) es una herramienta completamente libre basada en la plataforma *Windows* de Microsoft, liberada por primera vez en el año 2000 y desarrollada por Activeworx. Está diseñada para la gestión de algunos componentes configurables de Snort y las reglas de detección.

IDSPM centra su control en la administración de reglas de varios equipos donde Snort se ejecuta y en la gestión de estas reglas de manera centralizada (Alfon, 2009). Una de sus características más destacadas es que brinda una cómoda interfaz de usuario para todo el proceso de configuración de los archivos de reglas, lo que significa que todo se realiza mediante la GUI que ofrece la herramienta. Posibilita además la actualización de los paquetes de reglas, así como la activación/desactivación de los mismos (McRee, 2007).

Posee un explorador que facilita la organización de las configuraciones por sensor y grupos de reglas.

No obstante a los aspectos positivos antes señalados, se han identificado elementos desfavorables respecto a IDSPM. En primer lugar, la herramienta solo está disponible para la plataforma *Windows*, no es de código abierto y su empleo está limitado en cierto sentido por ser una aplicación de escritorio. La principal desventaja constituye que aparentemente el proyecto IDSPM se encuentra descontinuado, es decir, es un proyecto "muerto" el cual ya no tiene soporte o actualizaciones. Sobre este planteamiento no se encontraron evidencias en ningún documento oficial; sin embargo se han hallado argumentos recientes de la desactivación del proyecto en foros[17], el sitio oficial de la compañía no está habilitado[18] y se encuentra inactivo hace más de 3 años.[19]

1.4.2 SNORBY 2.5.6

Snorby es una aplicación web gratuita y de código abierto construida con el marco de trabajo *Ruby on Rails*[20] (RoR por sus sigla en inglés) para el monitoreo de seguridad en redes, el cual puede servir de interfaz con Snort y otros IDS como Suricata y Sagan. Está basado en el concepto de seguridad, sencillez y organización de la información (StaridsLabs, 2012). La interfaz de Snorby presenta una amplia variedad de vistas organizadas por categorías y gráficos, estructuradas de modo que su empleo puede realizarse de manera relativamente intuitiva.

Esta herramienta muestra un activo crecimiento por su comunidad de desarrollo. Puede instalarse en los teléfonos celulares inteligentes (*smartphones*) logrando hacer portable el monitoreo de la red. Respecto a las reglas, Snorby puede mostrar un gráfico de pastel reflejando la distribución de las mismas por categoría, informando incluso el número de paquetes analizados que han coincidido con cada tipo. Brinda un *dashboard*[21] en el que se integra parte de la información y permite la búsqueda de eventos por criterios (Community of Snorby, 2013). De la misma manera, puede ser desplegado en servidores como Apache

[17] http://blog.snort.org/2011/01/guis-for-snort.htm y https://www.alienvault.com/forum/index.php?t=msg&th=3154&start=0&S=af1c44dd958d605e903596dcbc8ece65
[18] http://www.activeworx.org/
[19] http://activeworx.blogspot.com/
[20] Es un marco de trabajo de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).
[21] Es una Interfaz gráfica de usuario que yace tanto en consolas de videojuegos como en algunos sistemas operativos y aplicaciones. Es una interfaz donde el usuario puede administrar el equipo y/o software.

y Nginx. Todo ello permite la centralización de la gestión de eventos de varios sensores. Además posibilita capturar completamente todo el tráfico de red.

Contrario a los elementos relacionados anteriormente, podemos decir que su poder de análisis y potencia en las funcionalidades no es muy alto (Alfons, 2010). Debido a que se centra en la gestión de alertas de Snort basado en sensores, casi no posee funcionalidades para la gestión de reglas y por el momento no permite la actualización de las mismas. Otro elemento desfavorable es que su *dashboard* solo es actualizado cada 30 minutos, y en ocasiones los gráficos generados pueden tener errores (StaridsLabs, 2012).

1.4.3 OSSIM 4.1

OSSIM es la abreviatura de Open Source Security Information Management System (en español sistema de código abierto para la gestión de la información de seguridad). Es una plataforma de seguridad desarrollada para gestionar la información de una red, analizarla y priorizar la información a mostrar. Es una distribución que integra más de 22 productos de seguridad (entre ellos Snort) todos ellos de código libre capaces de correlacionar la información generada por todos los componentes (Puchades Olmos, 2008). La plataforma puede organizar toda esta información basándose en el concepto de correlación.

De acuerdo con lo establecido en la fuente (López, 2011), la correlación es la posibilidad de obtener una visibilidad de todos los eventos de los sistemas en un punto y con un mismo formato, a través de esta situación relacionar y procesar la información para detectar, monitorizar, priorizar y organizar el estado de la seguridad de la red. OSSIM brinda la posibilidad de interrelacionar todas las funcionalidades de sus componentes.

OSSIM es una plataforma compleja pero a su vez potente, integrando soluciones de seguridad en una arquitectura abierta que se aprovechará de todas sus capacidades para aumentar la seguridad en las redes. Es libre y está disponible para Linux y Windows. En correspondencia con los datos del informe (Párrigas, 2010), OSSIM tiene la capacidad de aumentar la fiabilidad y la sensibilidad de los sensores que el sistema emplea. Funciona con múltiples monitores, puede inventariar activos de una red (IP, sistemas operativos, valor del activo) ante la falta de un administrador; lo que permite a su vez un mayor dominio sobre los componentes de la red. Además de generar informes, puede concebir tendencias de seguridad a lo largo del tiempo de ejecución en un ambiente monitorizado. Además puede integrarse con cualquier aplicación que envíe información a un servidor remoto.

Debido a la complejidad de la plataforma, realizar su configuración inicial no suele ser una tarea sencilla, y en adición se deben tener conocimientos avanzados de los componentes configurados. Respecto a las reglas empleadas por Snort, la plataforma se limita a emplear una vista por categorías de las mismas; no brinda ninguna mejora en sí, respecto a la gestión, edición y creación semiautomatizada o asistida con respecto a las reglas de detección (OSSIM, 2013). Adicionalmente, la integración de diferentes productos y servicios en OSSIM pueden ser vistos como una desventaja, pues al tratar de cubrir muchos aspectos, pudiera perderse el objetivo principal del administrador; conjuntamente con esto, pueden darse fallas en el sistema por un componente y ser un tanto difícil detectar qué componente causa la falla. En términos de costo la plataforma es completamente gratuita, pero su mantenimiento y soporte pueden ser caros (Nguyen, 2007).

1.4.4 OTRAS HERRAMIENTAS

Cabe mencionar la existencia de Oinkmaster y Pulledpork: unos *script* escritos en Perl[22] ampliamente usados para la actualización de paquetes de reglas.

Oinkmaster es un *script* Perl liberado bajo la licencia BSD y de código abierto, pues puede ser ejecutado tanto en sistemas Unix como en Windows. Permite mantener las reglas de Snort actualizadas descargando su código fuente, sin o con muy poca intervención del usuario. Puede asimilar reglas del sitio oficial de Snort, reglas de la comunidad o de terceras aplicaciones como *Bleeding Snort Rules*. De la misma manera contiene diversas funcionalidades que facilitan la gestión de reglas hasta cierto punto (Oinkmaster, 2013).

Pulledpork es un administrador de código abierto que automatiza la actualización de paquetes de reglas de Snort, mediante la descarga de estos teniendo como origen diferentes fuentes. Entre las funcionalidades que brinda están la verificación de los paquetes, la modificación del estado de una regla habilitándola o deshabilitándola, la gestión de los identificadores de reglas duplicados y el manejo de reglas compartidas (Pulledpork, 2013).

1.4.5 RESULTADOS DEL ESTUDIO

Luego de realizar un análisis de algunos sistemas con propósitos similares y GUIs de mayor popularidad para Snort, se evidencian puntos comunes referentes a la información relativa

[22] Perl es un lenguaje de programación que toma características del lenguaje C, del lenguaje interpretado bourne shell (sh), Lisp y, en un grado inferior, de otros lenguajes de programación. Estructuralmente, Perl está basado en un estilo de bloques y fue ampliamente adoptado por su destreza en el procesado de texto.

a las reglas de detección. Sin embargo es notable la diferencia en cuanto a las posibilidades que brindan y las funciones relativas a la gestión de reglas. Estas se limitan a brindar una lista de las reglas o una vista de su distribución. Los *scripts* analizados, de manera general solo ofrecen una automatización en la actualización de las reglas. Ninguna de las aplicaciones estudiadas mejora o asiste la gestión de reglas en cuanto a operaciones básicas se refiere; entiéndase la creación, eliminación modificación y otros eventos asociados, sin que eso conlleve la manipulación directa de los archivos de reglas; con excepción de IDS Policy Manager.

Como resultado se determinó que ninguna de las soluciones estudiadas satisface las necesidades de la investigación, debido a que no resuelven las dificultades planteadas en el problema de investigación. La herramienta que más se adapta a los objetivos es IDSPM 3 pero solo funciona sobre la plataforma Windows (hasta la versión Microsoft Windows 7), se encuentra descontinuada y fuera de desarrollo, por lo que no es recomendable su aplicación. No obstante, se identificaron aspectos positivos en los sistemas estudiados como son la organización de ficheros y reglas en árbol así como la centralización de la gestión de estas reglas para un ambiente distribuido, elementos incorporados en el desarrollo de la solución propuesta.

1.5 IDENTIFICACIÓN DE LAS HERRAMIENTAS Y TECNOLOGÍAS EMPLEADAS EN LA CONSTRUCCIÓN DE LA APLICACIÓN

En el proceso de desarrollo de *software* es de vital importancia la selección de las tecnologías y herramientas necesarias para la construcción del sistema. Cuando esta se hace de manera correcta, aumentan las probabilidades de éxito en la solución final. Igualmente, la selección acertada de una metodología de desarrollo de *software* afín a las necesidades y características del equipo de desarrollo, establece una guía por etapas en el proceso de construcción y garantiza en cierta medida la calidad y el cumplimiento de las expectativas. A continuación se expone la fundamentación de todos los elementos seleccionados para la implementación y diseño del módulo de gestión de reglas de detección del IDS Snort.

1.5.1 HERRAMIENTAS DE DISEÑO Y METODOLOGÍAS DE DESARROLLO

Una metodología de desarrollo de *software* se refiere a un marco de trabajo empleado para estructurar, planificar, documentar y guiar la construcción de aplicaciones informáticas. Se define según las etapas a seguir para el cumplimiento de un objetivo, en este caso: producir

un producto con la calidad requerida que cumpla con los requerimientos establecidos (Mendoza Sanchez, 2004). Sin una guía de este tipo, es posible en gran medida, que se obtenga un producto final que no era el esperado; lo que traería como consecuencia clientes insatisfechos con los resultados obtenidos.

Existen dos tipos de metodologías: robustas y ágiles. Las metodologías ágiles surgen como ramificación de las tradicionales, como alternativas más flexibles y adaptables a situaciones y características específicas del equipo de desarrollo. De igual modo, estas metodologías combinan una filosofía basada en la satisfacción del cliente y la entrega temprana de *software*, ajuste acertado en equipos pequeños y con alta motivación. De la misma manera, estas técnicas definen un conjunto de directrices fundamentadas sobre el análisis, el diseño y la comunicación activa y continua entre el equipo de desarrollo y el cliente. Estos tipos de metodologías han demostrado su utilidad al entregar sistemas exitosos con rapidez (Pressman, 2010).

Programación Extrema (*Extreme Programing*, XP por sus siglas en inglés) es una metodología de desarrollo de software de gran utilidad en proyectos de corto plazo, equipos pequeños y donde es necesario una entrega veloz. De acuerdo con lo expresado en algunas bibliografías (Mendoza Sanchez, 2004), la filosofía de XP consiste en una programación rápida, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. De manera básica dicha metodología se fundamenta sobre la base de la simplicidad, la comunicación y reutilización del código desarrollado; así como la fomentación de la retroalimentación concreta y frecuente del equipo de desarrollo, cliente y los usuarios finales.

El diseño de la misma está pensado para adaptarse con muy poco costo a los cambios frecuentes suscitados en el proceso de desarrollo, debido al diseño de su ciclo; aspecto positivo de XP en comparación con metodologías que siguen un modelo de desarrollo de tipo cascada e iterativas. Su continua evolución y progreso permiten un desarrollo adaptable y ágil. Esa característica permite que el costo del cambio no dependa de la fase o etapa correspondiente y tenga un impacto realmente pequeño. Lo anterior es ilustrado en la Figura 6.

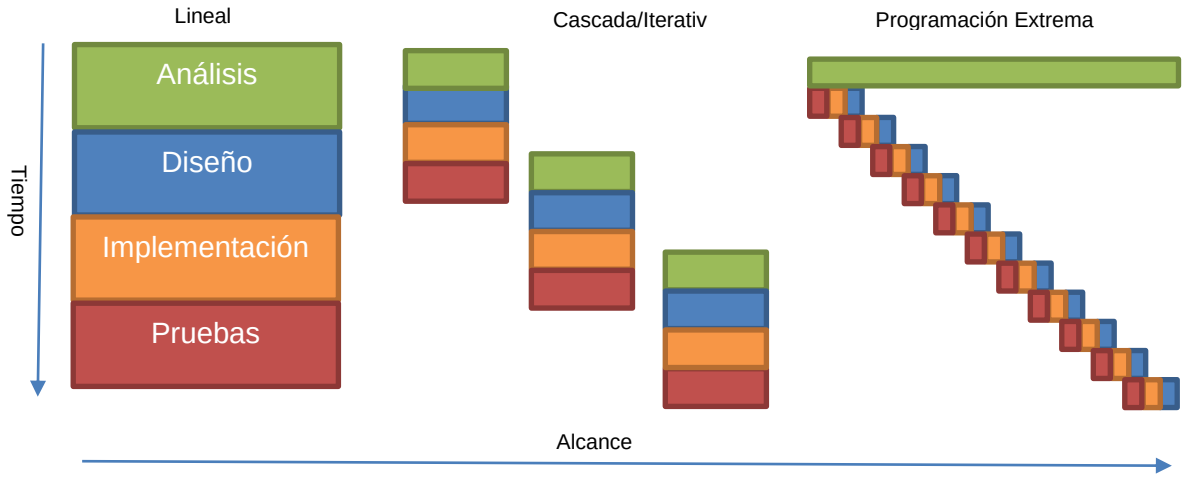


Figura 6 Comparación de la adaptabilidad al cambio y el costo en tiempo de XP con respecto a otros tipos de enfoque (Erlijman Piwen, y otros, 2001).

El ciclo de XP se compone de cuatro etapas de manera parcialmente repetitiva: Planificación, Diseño, Desarrollo y Pruebas. Dentro de los elementos fundamentales que implementa se encuentran (Letelier, y otros, 2009):

- **Entregas pequeñas:** producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Estas versiones constituyen resultados de valor para el negocio. Una entrega pequeña no debería tardar más de 3 meses.
- **Pruebas Unitarias:** pruebas realizadas a los principales procesos, de tal manera en etapas tempranas, se pueden realizar pruebas de las fallas que pudieran ocurrir. De este modo es posible avizorar los errores que pueden aparecer.
- **Programación en parejas:** toda la producción de código debe realizarse en parejas de programadores, según un estudio realizado para identificar los costos y beneficios de la programación en parejas (Cockbun, y otros, 2000). Esto conlleva ventajas implícitas: menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, entre otras.

- **Cliente in-situ:** el cliente tiene que estar presente y disponible todo el tiempo para el proceso de construcción. Éste es uno de los principales factores de éxito de la metodología XP. El cliente conduce constantemente el trabajo lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada.

En contraste con todos los elementos expuestos, se plantea que debido al pequeño tamaño del equipo de desarrollo, la cantidad de cambios que se avizoran en el proceso de construcción, la necesidad de adaptabilidad y flexibilidad durante el mismo y el contacto directo con el cliente (en este caso el departamento de seguridad digital del Centro de Identificación y Seguridad Digital de la UCI), conjuntamente con la necesidad de agilizar el proceso de construcción de la solución y la característica de tener al cliente con participación activa en todo el ciclo de vida del sistema, se identifica XP como metodología de desarrollo de *software* para la construcción de la aplicación.

Para el modelado se empleó el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) es un lenguaje muy conocido y utilizado en la actualidad para esquematizar sistemas de software, respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico con el fin de visualizar, especificar, construir y documentar un sistema (OMG, 1997). Para el desarrollo de la aplicación se utilizará UML en su versión 2.0 en la confección de los elementos de diseño. Se elige este lenguaje principalmente por su gran popularidad de uso y aceptación en el desarrollo de software.

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE por su siglas en inglés) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de aplicaciones reduciendo el costo en términos de tiempo y de dinero. Estas herramientas facilitan muchos aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

Como herramienta CASE se eligió *Visual Paradigm for UML 10.1 Standard Edition* usada en su versión libre, la cual brinda una serie de funcionalidades basadas en el leguaje de modelado UML. Soporta muchos estándares de modelado como UML, diagramas de relación de entidades (diseño de bases de datos), modelado de procesos de negocio (flujo de datos del negocio y diagramas de proceso de negocio), arquitectura empresarial (diseño

de componentes empresariales), entre otros. Adicionalmente proporciona soporte a los equipos de desarrollo en el planeamiento de *software,* ingeniería de código (convierte código fuente en diagramas y viceversa incluyendo a código Python), modelados de clases y modelado de datos. Por ello sustenta el ciclo de vida completo del desarrollo de *software* (VP Inc., 2013). En la generación computacional de los diagramas de diseño y modelación de los componentes del sistema propuesto se empleó esta herramienta CASE.

1.5.2 TECNOLOGÍAS PARA EL DESARROLLO DE LA LÓGICA DE NEGOCIO

Las tecnologías son esenciales para cualquier tipo de proceso de desarrollo de software. El equipo de trabajo se vale de estas para construir la solución que planifican y diseñan previamente. En consecuencia, es muy importante realizar una buena selección, pues de la relación de los elementos a utilizar depende en cierta medida la calidad del producto final. Igualmente, la habilidad que tenga el equipo en el manejo de dichas tecnologías influye de manera positiva en el empleo de sus ventajas, el cumplimiento de los objetivos trazados, la eficiencia y eficacia con que estos logran los objetivos propuestos. A continuación se describen las tecnologías elegidas en la presente investigación.

Python es un lenguaje de programación de alto nivel, versátil y con una sintaxis simple que favorece un código legible y es utilizado actualmente en una amplia variedad de dominios de aplicación (Python Software Fundation, 2013). Se trata de un lenguaje interpretado, con tipado dinámico[23], fuertemente tipado[24], orientado a objetos y multiplataforma, característica (esta última) que lo dota de portabilidad. Necesita de un intérprete para que pueda ser ejecutado, no obstante tiene muchas características de los lenguajes compilados por lo que se podría decir que es semi-interpretado. Por ejemplo, el código fuente es traducido a un pseudo código máquina intermedio llamado *bytecode* la primera vez que se ejecuta; generando archivos de formato *.py* o *.pyc* los cuales son *bytecode* optimizado y son lo que se ejecutarán en sucesivas ocasiones. No obstante, en algunas bibliografías (Gonzales Duque, 2010) se explica que python no es adecuado para aplicaciones de bajo nivel o donde el rendimiento sea crítico, debido a que puede ser algo lento por su característica de ser interpretado.

[23] En un lenguaje dinámicamente tipado, no suele ser necesario declarar el tipo de las variables. Es típico de los intérpretes como PHP o Python. Las variables empiezan a existir cuando se les da valor y estos tienen en cuenta los tipos, desde luego, pero no de las variables, sino de su contenido.

[24] Un lenguaje de programación es fuertemente tipado si no se permiten violaciones de los tipos de datos, es decir, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión. No hay una única definición de este término.

Python reconoce la programación imperativa y la programación funcional, brindando amplias posibilidades en la escritura del programa. Además, estructura su código de manera modular permitiendo añadir e integrar componentes de forma simple. Esto posibilita que cuente con extensas bibliotecas de código estándar y de terceros para prácticamente cualquier tarea. Se requiere adicionalmente, que la aplicación desarrollada pueda ejecutarse en múltiples plataformas lo cual estaría resguardado con la utilización de Python.

El lenguaje en sí presenta potencia y flexibilidad, que le permite al programador manejar prácticamente cualquier domino de problema. Se integra muy bien con otras plataformas como .Net. Adicionalmente, Python está bajo una licencia de código abierto (*Python Software Foundation License Agreement*) y es libremente usable y distribuible incluso para uso comercial. Se utiliza en el presente desarrollo la versión 2.7, la cual en estos momentos es la más reciente que puede integrarse con el marco de trabajo seleccionado (Python Software Fundation, 2013). Presenta una sintaxis sencilla y se adapta de manera práctica al domino de problema de la presente investigación.

Un marco de trabajo o *framework*, provee al desarrollador una infraestructura de programación. Se refiere a una estructura compuesta de componentes personalizables e intercambiables para el desarrollo de un sistema (Holovaty, y otros, 2009). De modo práctico, consigue considerarse como una aplicación genérica y configurable a la que se le pueden añadir las piezas para construir la aplicación concreta. En concordancia el portal oficial de Django (Django Software Fundation, 2013) este se define como un marco de trabajo de desarrollo web de código abierto escrito en Python. Se basa en el desarrollo rápido, el diseño limpio y pragmático[25], tratando de automatizar todo lo posible la construcción de una aplicación web y se adhiere al principio DRY (don'y repeat yourself, traducido al español literalmente como: no te repitas).

Este marco de trabajo se plantea como objetivo primordial facilitar la creación de sitios web complejos, poniendo énfasis en la reutilización, la conectividad, el desarrollo ágil y la extensibilidad de componentes. Cabe mencionar otras posibilidades que permite como: mapeo de objetos relacionales, interfaz de administración automática, diseño de URL limpias[26], sistemas de plantillas, sistema de caché, y soporta aplicaciones multilenguaje estableciendo cadenas de traducción (Django Software Fundation, 2013). Python es

[25] *adj.* Referente a la acción y no a la especulación, relacionado con las el valor práctico de las cosas.
[26] Son aquellas URLs que son, dentro de lo que cabe, entendibles para el usuario. No contienen variables ni caracteres extraños.

utilizado en todos los elementos del marco de trabajo, incluso en configuraciones, archivos, estructuras y en los modelos de datos.

Conjuntamente con todas las premisas expuestas, se pretende que la aplicación a construir sea lo más desacoplada posible para garantizar la futura extensibilidad, reutilización y fácil mantenimiento de la misma, por ello se recurre a Django en su versión 1.4.1. Pues por defecto contiene muchas funciones útiles para tareas comunes del desarrollo web, como la autenticación de usuarios o la interfaz de administración de contenido, ahorrando tiempo de construcción. Además, presenta una comunidad amplia con disímiles aplicaciones de terceros.

1.5.3 TECNOLOGÍAS PARA EL DESARROLLO DE LAS INTERFACES DE USUARIO

El desarrollo de la interfaz de usuario es una pieza importante en la implementación del sistema. La interfaz de usuario es el medio mediante el cual los usuarios pueden interactuar con el sistema. Se debe caracterizar por su fácil comprensión y simpleza en el diseño. A continuación se describen las tecnologías empleadas para su construcción.

El Lenguaje de Marcado de Hipertexto (*Hyper-Text Markup Language*, HTML por sus siglas en inglés) con el cual se construyen las páginas web que interpretan los navegadores web, las cuales no son más que documentos de hipertexto. De modo básico, el lenguaje constituye un conjunto de etiquetas de marcado que sirven para definir cómo se muestra el texto y otros elementos que compondrán una página. Una combinación de estas etiquetas, agrupadas y anidadas, constituye la estructura del documento, estas le indican al navegador web cómo mostrar todos los contenidos de la página, además de que complementa la información de un documento con otros elementos como las imágenes.

Hasta cierto punto HTML puede representar la apariencia de un documento y puede embeber código en otros lenguajes como *JavaScript* para agregar elementos y modificar el comportamiento del documento web (Raggett, 2003). En el desarrollo se utilizó su versión 4.0 presente en todas las plantillas de la aplicación. El lenguaje HTML por sí solo limita el modo en que puede mostrar sus elementos. Tiene buen desempeño organizando o describiendo contenido como párrafos o textos, pero la apariencia de las páginas creadas sólo con HTML es bastante rústica. CSS significa Hojas de Estilo en Cascada, por sus siglas en inglés, enfocadas a proporcionar un diseño admisible en la presentación de los documentos que contienen HTML.

Es un lenguaje que describe la manera en que son mostrados los documentos HTML, estructurados en hojas de estilo ofreciendo gran flexibilidad en el diseño de sitios y aplicaciones web. Permite la separación de la presentación y el contenido. De manera equivalente facilita la unificación de la apariencia, donde una sola hoja de estilo puede dar formato al contenido de todas las páginas del espacio web; lo que permite hacer cambios en un solo lugar para modificar la apariencia del sitio (Mansfield, 2005).

En la construcción de la solución se eligió CSS en su versión más reciente 3.0, la cual brinda un control más abarcador sobre los elementos del documento (De Luca, 2010), incorporando nuevos mecanismos sobre la manera en que los elementos de la página son mostrados, sin tener que recurrir a trucos necesarios en las versiones anteriores del estilo en cascada. Este lenguaje permitió enriquecer gráficamente la GUI del sistema otorgándole un mejor acabado.

Otra tecnología empleada en la construcción de la interfaz es JavaScript (JS). Este un lenguaje de programación del lado del cliente (*client-side*)[27], ligero y libre, diseñado para añadir interactividad y dinamismo a las páginas web, a la vez que permite realizar parte del procesamiento del lado del cliente, lo que reduce la carga de trabajo en el servidor. Es una implementación del estándar ECMA *script* (ECMA-262 estándar oficial) desarrollado y mantenido por la organización *ECMA International*. JS está orientado a eventos con manejo de objetos, por lo que se puede aplicar la programación orientada a objetos. Es un lenguaje interpretado, que tiene como intérprete al mismo navegador web (Álvarez, 2001). Lo anterior posibilita que sea un lenguaje que se pueden emplear independientemente de la plataforma usada, dotando de estas características a las aplicaciones que son escritas con este.

Su empleo permite crear contenidos dinámicos y posibilita realizar validaciones del contenido antes de ser enviado al servidor, aligerando la carga de este último. Es un lenguaje ampliamente usado hoy en día y cuenta con disimiles librerías de códigos que aumentan su usabilidad, manteniendo el mismo fin y mejorando su empleo, elementos que proporcionan gran usabilidad (Bradenbaugh, 2000). Se empleó JavaScript 1.8.5, integrado con HTML y CSS, elementos ampliamente empleados hoy en día en el desarrollo de aplicaciones web, posibilitando incorporar las ventajas antes descritas.

[27] El navegador web es el que se ocupa del procesar el lenguaje, no el servidor desde donde se envía la página. Por ello es un lenguaje del lado del cliente referente a la arquitectura cliente-servidor (ver glosario de términos).

La interacción e intercambio de información entre los componentes del servidor y el cliente se logran con el empleo de JSON (*Java Script Object Notation*, en español Notación de objetos de JavaScript), el cual constituye un formato ligero de intercambio de datos. Leerlo y escribirlo es simple y la manera de procesarlo y generarlo también lo es. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edición- Diciembre 1999 (ECMA Script Group, 2013). Es un formato de texto completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos entre el cliente y el servidor.

CoffeeScript[28] (CS) es un lenguaje pequeño que permite traducir su código a JavaScript. Su sintaxis está inspirada en el lenguaje Ruby y Python e implementa varias características de esos lenguajes. CS resume el código que se necesita escribir en comparación con JS, ya que es muy breve en cuanto a su sintaxis; reduciendo a la mitad o en ocasiones hasta la tercera parte de su traducción al código JavaScript. Igualmente puede emplearse integrado a cualquier librería JS (MacCaw, 2012).

Debido a que los componentes JS de la aplicación tienen una complejidad considerable, y el código del lado del servidor se escribe con Python, se elige CS para hacer más amena la implementación de dichos componentes, debido principalmente a la similitud de sintaxis con Python. Además de que a causa de la reducción del código, la implementación es más rápida. La versión 1.6.2 es la más reciente y fue la utilizada en la implementación de la aplicación. El uso de esta tecnología en la aplicación está extendido a todos los componentes de la GUI, igualmente se manifiesta en componentes lógicos que son ejecutados en el cliente como el intérprete de las reglas y sus unidades.

Otra tecnología empleada es ExtJS la cual constituye una librería JavaScript que viabiliza construir, relativamente fácil, una interfaz de usuario con estilo similar al de una aplicación de escritorio en un ambiente web. Permite la manipulación de componentes, como son: múltiples ventanas, barras de herramientas, menús desplegables, cuadros de diálogo y otros (Shea, y otros, 2010) que se asemejan a los ambientes de escritorio. Conjuntamente con lo anterior provee además un marco de trabajo MVC orientado a objetos y extensible (Fronckowiak, 2008).

[28] También llamado CS, por sus siglas en inglés.

Se aprovecha esta librería en su versión 4.1.1 debido a la posibilidad que presenta de proveer un marco de trabajo para estructurar los componentes JavaScript en su totalidad; muy conveniente en el desarrollo de la presente aplicación por la complejidad que estos alcanzan en el sistema. Además, posibilita la construcción de la interfaz sencilla y amigable que se tiene como objetivo en la presente investigación. La interacción con CoffeeScript se logró satisfactoriamente en el proceso de construcción.

1.5.4 ENTORNO DE DESARROLLO INTEGRADO

Un IDE, acrónimo de *Integrated Development Environment* (en español Entorno de Desarrollo Integrado) es un sistema que facilita el trabajo del desarrollador de software, integrando sólidamente la edición orientada al lenguaje, la compilación o interpretación, la depuración, las medidas de rendimiento y más funcionalidades en dependencia del IDE en cuestión; normalmente presentando lo anterior de forma modular y haciéndolo extensible. Es posible además que un mismo IDE funcione con varios lenguajes de programación (Blogsfarm Network SL , 2011). Estos pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Pycharm es un IDE propietario de JetBrains. Este entorno de desarrollo es diseñado específicamente para integrar Python y Django, soportando el desarrollo web integrando también HTML y JavaScript. Contiene un editor de texto inteligente lo que permite la codificación cómoda al desarrollador y varias utilidades como en el resaltado de sintaxis erróneas, auto identación, formateo y completamiento de código, así como el archivamiento del código escrito, entre otros (PR Newswire Association LLC., 2013). Permite navegar a través del código fuente y realizar búsquedas, a pesar de que dicho código puede crecer considerablemente. Igualmente, posibilita la refactorización y la documentación del texto escrito.

Además, Permite la integración con un sistema de control de versiones, así como con muchos módulos disponibles, los cuales posibilitan diversas funcionalidades, extendiendo las del propio IDE (JetBrains, 2013). La UCI brinda una licencia legítima y gratuita para uso de la comunidad universitaria, empleada durante el proceso de desarrollo con Pycharm en su versión 2.6.3.

Capítulo 2 Planificación y Diseño del Módulo de Gestión de Reglas de Detección

La metodología XP define que el proceso de desarrollo debe estar guiado por las historias de usuario, pruebas automatizadas, una metáfora del sistema y una planificación flexible que organice dicho proceso. En el presente capítulo se describe la planificación y diseño vinculando todos los elementos anteriores. De manera explícita se describen las necesidades del usuario, la organización del desarrollo y las entregas pequeñas. Adicionalmente, se reflejan los patrones arquitectónicos, patrones de diseño y tarjetas CRC estructurando de esta manera los componentes que conforman la maquetación del sistema.

2.1 ANÁLISIS DEL PROBLEMA

Luego de instalar en una plataforma determinada el sistema de detección de intrusos Snort, es recomendada la instalación de una herramienta de administración como las presentadas en el epígrafe 1.4. Estas se utilizan en la supervisión de la información generada por el IDS. A pesar de ello, la configuración y puesta a punto del sistema de detección implica en la mayoría de los casos la manipulación directa de los archivos de configuración, lo mismo ocurre con las reglas de detección. Estas se almacenan en archivos clasificados según el tipo de ataque, sistema operativo, tipo de amenazas, tipo de servicios o aplicaciones que corren en la red o cualquier otro criterio que se definan. No obstante, esta clasificación sigue respondiendo de forma más específica a las cuatro categorías mencionadas con anterioridad.

Para activar, añadir o modificar una o varias reglas habría que realizar la operación de forma manual directamente en el fichero, aumentando la posibilidad de ocasionar errores por un manejo incorrecto del archivo convirtiéndose en una tarea repetitiva que se debe realizar cada vez que se necesita variar la configuración. Proceder de esta manera puede incidir en el mal funcionamiento del IDS. Igualmente, este proceder es algo engorroso debido a que una regla puede tener una estructura compleja, pues solo un usuario con experiencia y conocimientos avanzados puede realizar esta tarea de manera efectiva. Por ello, los desarrolladores de Snort recomiendan realizar un proceso de refinado de las reglas (Roesch, y otros, 2012), con el objetivo de incidir de manera positiva en la configuración de estas, equilibrando el procesamiento y disminuyendo falsos positivos. Esto está dado por el hecho de la relación existente entre el funcionamiento del motor de detección y la configuración de las reglas.

La situación descrita anteriormente no agiliza el tratamiento de los ficheros, ni ayuda a minimizar los errores lógicos o de escritura al construir una regla. Además la cantidad de reglas existentes por defecto es elevada y tienden a incrementar con el tiempo. En consecuencia, no se cuenta con un modo idóneo para la gestión de reglas de detección de Snort; lo que provoca además un incremento de la posibilidad de que se cometan errores por los administradores, afectando el buen funcionamiento del IDS. Todas las consideraciones anteriormente expuestas, pretenden ser resueltas mediante la construcción del Módulo de gestión de reglas de detección.

2.2 HISTORIAS DE USUARIO

El primer artefacto generado en la etapa inicial de planificación de la metodología XP son las historias de usuario (HU), constituyendo los primeros pasos para la conformación del proyecto. Las HU son una técnica para capturar los requerimientos de un sistema, las cuales son escritas por el cliente empleando el lenguaje común, como descripciones escuetas de lo que el sistema debe realizar; sin embargo el equipo de desarrollo también puede escribir historias de usuario (Jeffries , y otros, 2001). En orden con la fuente (Tangient, 2013), el sistema a desarrollar es descrito completamente por las historias de usuario; pues las HU son una forma rápida de administrar los requisitos sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. De la misma manera, las HU se emplean en la fase de pruebas para verificar si el programa cumple con las especificaciones del cliente.

El tratamiento de las historias de usuario es un proceso dinámico y flexible; en cualquier momento estas pueden ser eliminadas, reemplazadas por otras más específicas o generales y se pueden añadir nuevas o modificar historias existentes. Este tratamiento permite responder rápidamente ante requisitos cambiantes y dividir el desarrollo de los proyectos en pequeñas entregas, lo cual viabiliza la estimación del esfuerzo de implementación (Jeffries , y otros, 2001). Respecto a la información contenida en la historia de usuario no existe un consenso general, no obstante Kent Beck en su libro (Beck, 2000) ofrece una sugerencia de los campos que una historia de usuario puede contener, explicando la no obligatoriedad de seguir su definición, por lo que esta puede ser adaptada a las necesidades de cada proyecto.

Las historias de usuario definidas por el cliente son las siguientes:

- HU 01 Construir árbol de ficheros y reglas.

- HU 02 Crear regla.
- HU 03 Modificar regla.
- HU 04 Eliminar regla.
- HU 05 Leer regla.
- HU 06 Listar reglas.
- HU 07 Cargar fichero de reglas.
- HU 08 Crear editor de reglas con codificación inteligente.

Historia de Usuario						**No.**	01
Nombre	Construir árbol de ficheros y reglas						
Usuario	Administrador						
Prioridad	alta	**Complejidad**	alta	**Estimación** (**sem**)	3	**Tipo**	funcional
Dependencia	HU 07, HU5						
Descripción							
El administrador desea contar con una estructura de árbol que muestre los ficheros y las reglas de manera jerárquica. Lo cual permite al administrador tener rápido acceso a los ficheros y su contenido.							
Información Adicional							
Al hacer clic con el botón izquierdo en un fichero de regla, el sistema despliega las reglas que contiene indexadas con respecto al fichero.							

Tabla 3 Descripción de la HU número 1

Historia de Usuario						**No.**	02
Nombre	Crear regla						
Usuario	Administrador						
Prioridad	alta	**Complejidad**	Media	**Estimación** (**sem**)	1	**Tipo**	funcional
Dependencia	HU 08						
Descripción							
El administrador desea crear una regla en un fichero previamente seleccionado escribiendo la misma en el editor de textos que el sistema incorpore. Esta regla debe agregarse a dicho fichero.							
Información Adicional							
El administrador confecciona las reglas con la ayuda del editor, por lo cual esta historia tiene cierta dependencia respecto a la HU 08.							

Tabla 4 Descripción de la HU número 02

Historia de Usuario						**No.**	08
Nombre	Crear editor de reglas con codificación inteligente						
Usuario	Administrador						
Prioridad	media	**Complejidad**	alta	**Estimación** (**sem**)	4	**Tipo**	funcional
Dependencia	-						

Descripción
El administrador desea contar con un editor que considerando el texto que ha escrito, ofrezca una asistencia dinámica en cuanto a completamiento de texto y resaltado de sintaxis incorrecta. Lo cual necesita incorporar en la creación y modificación de una regla de detección.
Información Adicional
Se necesita una ubicación válida respecto al fichero de configuración de Snort.

Tabla 5 Descripción de la HU número 08

El resto de las historias de usuario se describen en los anexos Historias de usuario.

Según el autor (Cohn, 2012), siguiendo la metodología XP los requerimientos no funcionales del software puede ser fácilmente gestionados como historias de usuario. Se consideró de esta manera dar tratamiento a este tipo de requerimiento. A continuación se listan las historias de usuario categorizadas por tipos no funcionales.

- Usabilidad:
  - HU 09 Orientación a íconos.
- Fiabilidad:
  - HU 10 Tiempo de reparación menor de 7 días.
  - HU 11 Disponibilidad de información y funcionalidades.
  - HU 12 No ocultación o rastreo.
- Restricción de diseño:
  - HU 13 Diseño de pocas entradas.
  - HU 14 Basado en arquitectura cliente-servidor.
- Portabilidad:
  - HU 15 Sistema multiplataforma.
- Legales:
  - HU 16 Licencias de herramientas.
  - HU 17 Normativas legales de la Universidad de Ciencias Informáticas.

Historia de Usuario						**No.**	12
Nombre	No ocultación o rastreo						
Usuario	Administrador						
Prioridad	baja	**Complejidad**	baja	**Estimación** (**sem**)	< 1	**Tipo**	propiedad
Dependencia	-						
Descripción							
Se consideró de esta manera dar tratamiento a este tipo de requerimiento. A continuación se listan las historias de usuario categorizadas por tipos no funcionales..							

Información Adicional
Tributa a la **fiabilidad** del sistema.

Tabla 6 Descripción de la HU número 12

Historia de Usuario						**No.**	15
Nombre	Sistema multiplataforma						
Usuario	Administrador						
Prioridad	media	**Complejidad**	media	**Estimación** (**sem**)	1	**Tipo**	propiedad
Dependencia	-						
Descripción							
Se requiere que el sistema pueda ejecutarse en varios sistemas operativos como Microsoft Windows, Linux y Mac.							
Información Adicional							
Tributa a la portabilidad del sistema.							

Tabla 7 Descripción de la HU número 15

Historia de Usuario						**No.**	16
Nombre	Licencias de herramientas						
Usuario	Administrador						
Prioridad	media	**Complejidad**	media	**Estimación** (**sem**)	< 1	**Tipo**	propiedad
Dependencia	-						
Descripción							
Se solicita que la mayoría de las herramientas empleadas en el desarrollo sean liberadas bajo licencias como GPL, LGPL, BSD, CDDL. En caso contrario contar con una licencia que avale el empleo de la herramienta.							
Información Adicional							
Corresponde a aspectos legales del sistema.							

Tabla 8 Descripción de la HU número 12

El resto de las historias de usuario se describen en los anexos Historias de usuario.

2.3 DESCRIPCIÓN DEL SISTEMA

El sistema propuesto para dar solución al problema descrito anteriormente, está enfocado a ofrecer principalmente la gestión de los archivos de las reglas de detección. Se pretende brindar un *front-end*[29] al usuario desde el cual se podrán realizar todas las operaciones que el sistema provee. En aras de organizar la información el sistema consta de dos partes principales. En una se muestran los archivos de reglas organizados y clasificados según su tipo, donde cada uno despliega las reglas que tiene asociadas. Lo anterior suministra una vista categorizada de los archivos de reglas, donde el usuario puede lograr una mayor

[29] Componente del software, sistema o aplicación que interactúa con el o los usuarios.

interacción con los elementos gestionados. Este componente realiza una búsqueda de los archivos de reglas en la ubicación del equipo donde Snort se ejecuta, colocándolos en el *front-end* para su visualización.

El segundo componente se basa en un editor que permitiendo la construcción o edición de una regla de detección. El editor ofrece el respaldo de una codificación inteligente que brinda asistencia al usuario en el resaltado de sintaxis incorrecta y completamiento de opciones en la codificación; sirviendo de guía en el proceso de construcción y edición de la regla. Los elementos descritos, componen en su conjunto las partes claves del sistema que se propone como solución.

2.4 PLANIFICACIÓN

En relación a los contenido mostrados en las fuentes (Beck, 2000) y (Bajo de Luque, 2011), se establece la fase de planificación como un elemento esencial para la organización del proceso de desarrollo de un proyecto. El objetivo de la planificación en XP se centra en que ambas partes del equipo de proyecto: clientes y desarrolladores acuerden una estimación de las historias de usuario más valiosas que serán implementas en primera instancia y las entregas pequeñas del producto asociadas a estas. Como parte fundamental de esta etapa, se estima el esfuerzo propuesto para la realización de cada HU y se procede a la planificación de la etapa de implementación del sistema. En este plan se especifica exactamente cuáles historias de usuario serán implementadas en cada iteración y las posibles fechas de entrega, considerando que esta última debe obtenerse en no más de tres meses.

La planificación puede ejecutarse en base al tiempo o al alcance. Según el autor (Joskowicz, 2008), al planificar por tiempo se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar en cuanto al alcance del sistema, se divide la suma de puntos de las historias seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación. Un punto equivale a una semana ideal (40 horas de trabajo según XP) de programación.

2.4.1 ESTIMACIÓN DE ESFUERZO POR HISTORIAS DE USUARIO

Las estimaciones de esfuerzo asociadas a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto equivale a una semana ideal (40 horas de trabajo según XP) de programación (Joskowicz, 2008). Por otra parte, el

equipo de trabajo mantiene un registro de la velocidad de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración realizada.

La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Solo se estimarán las historias de usuario que corresponden a funcionalidades del sistema, debido a que las restantes (de la HU9 a la HU 17) responden a propiedades que no conllevan a un esfuerzo notable en la implementación de la aplicación. La Tabla 9 muestra la estimación del esfuerzo de implementación por HU.

No.	Historia de Usuario	Estimación (sem)
01	Construir árbol de ficheros y reglas	**2**
02	Crear regla	**1**
03	Modificar regla	**1**
04	Eliminar regla	**0.5**
05	Leer regla	**2**
06	Listar reglas	**0.5**
07	Cargar fichero de regla	**1**
08	Crear editor de reglas con codificación inteligente	**4**
	TOTAL	**12**

Tabla 9 Estimación de esfuerzo por historia de usuario

2.4.2 PLAN DE ENTREGA

El plan de entrega establece qué historias de usuario serán agrupadas para conformar una entrega y el orden de las mismas. Este plan es confeccionado entre clientes y el equipo de desarrollo. De acuerdo a lo establecido por el autor (Beck, y otros, 2000), para la realización de este plan, el cliente ordena y agrupa según sus prioridades las historias de usuario; no obstante, para conformar el plan de entrega, el orden establecido por el cliente se contrasta con las estimaciones de tiempo de desarrollo realizadas por los desarrolladores y sus criterios en cuanto a la secuencia de implementación de las HU.

Luego de algunas iteraciones, es prudente realizar nuevamente una reunión con los actores del proyecto para evaluar el plan de entrega y ajustarlo si es necesario, pues es probable que surjan cambios en el desarrollo (Joskowicz, 2008); consecuentemente este plan debe ser revisado con frecuencia, por ello se considera sólo una instantánea de qué elementos se implementarán y cuándo. Lo anterior ayuda a que todo el equipo de trabajo tenga una idea de la planificación a grandes rasgos y de forma no determinante (Beck, y otros, 2000).

A continuación se definen dos tipos de planes de entregas: uno en función de las historias de usuario y el otro resume la información del primero definiendo las fechas de las entregas pequeñas del producto.

Orden	No.	Historia de Usuario	Estimación	Iteración Asignada	Entrega Asignada
1	**05**	Leer reglas	**1**	**1**	**1**
2	**07**	Cargar fichero de reglas	**1**	**1**	**1**
3	**01**	Construir árbol de ficheros y reglas	**3**	**1**	**1**
4	**06**	Listar reglas	**0.5**	**1**	**1**
5	**08**	Crear editor de reglas con codificación inteligente	**4**	**2**	**1**
6	**02**	Crear regla	**1**	**3**	**2**
7	**03**	Modificar regla	**1**	**3**	**2**
8	**04**	Eliminar regla	**0.5**	**3**	**2**

Tabla 10 Plan de entrega en función de las HU

Entrega	Fecha
Inicio de Implet.	18 febrero 2013
Primera Entrega	28 abril 2013
Segunda Entrega	18 de junio 2013

Tabla 11 Plan de entrega resumido con fechas

2.4.3 PLAN DE ITERACIONES

Mientras que el plan de entrega está sincronizado al ritmo del negocio, el plan de iteraciones se sincroniza al ritmo de la implementación del sistema, de aquí que este sea confeccionado por los programadores en gran medida; basándose para ello en criterios aterrizados en la construcción de las HU y su agrupación en iteraciones. No obstante el cliente puede participar activamente según la fuente (Beck, y otros, 2000). Las historias de usuario seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Luego, quedan definidas las iteraciones a la que corresponde la implementación de cada HU y la estimación del tiempo total de cada iteración.

Iteración	HU	Orden de implementación de HU	Duración Total (sem)
1	05	Leer reglas	**5.5**
	07	Cargar fichero de reglas	
	01	Construir árbol de ficheros y reglas	
	06	Listar reglas	

2	08	Crear editor de reglas con codificación inteligente	**4**
3	02	Crear regla	**2.5**
	03	Modificar reglas	
	04	Eliminar regla	

Tabla 12 Plan de iteraciones

2.4.4 TAREAS DE INGENIERÍA

Las HU que corresponden a cada iteración se desglosan en tareas que contribuyen a su realización. Estas son definidas por los desarrolladores y no son de interés del cliente. Cada programador se hace responsable de una o más tareas, además de que estas pueden estar compartidas entre varios programadores (Beck, y otros, 2000). Esto se realiza con el objetivo de contar con unidades pequeñas que respondan a las historias de usuario para realizar un seguimiento de estas últimas. Estas tareas generalmente se definen mediante tormentas de ideas llevadas a cabo entre los desarrolladores. De manera objetiva, se muestra en la Tabla 13 la asociación de las tareas respecto a las historias de usuario.

HU	No.	Tarea	Estimación(días)
05	**01**	Implementar método de búsqueda sobre ficheros	**1**
	02	Definir forma de leer fichero de reglas con desplazamiento sobre este	**2**
	03	Delimitar una regla de otra en el proceso de lectura	**3**
	04	Leer regla completa del fichero (una)	**3**
07	**05**	Establecer clase en el modelo que defina una regla del fichero	**2**
	06	Manejar las colecciones de la clase regla	**3**
01	**07**	Definir librería JavaScript para la implementación	**3**
	08	Capturar eventos de clic sobre ficheros y reglas mostrados	**3**
	09	Crear recorrido recursivo de ficheros y directorios	**3**
06	**10**	Mostrar una lista de reglas cargadas de un fichero	**1**
08	**11**	Crear un autómata que represente sintaxis para crear regla	**10**
	12	Definir gramática	**4**
	13	Implementar *scaner*	**5**
	14	Implementar *pharser*	**5**
02	**15**	Escribir texto de editor en fichero de regla	**3**
03	**16**	Remplazar regla por texto de editor	**4**
04	**17**	Desactivar la regla	**3**

Tabla 13 Resumen de Tareas de Ingeniería

2.5 DISEÑO

El diseño tiene un rol esencial en el desarrollo de software. A partir de este se establece una plataforma que soporta la construcción de un sistema computacional. Como resultado, se definen varios modelos del sistema a construir, estableciendo una especie de plan para producir la aplicación (Billy, 2013). Estos modelos pueden evaluarse en relación con su calidad y mejorarse antes de generar código, pues constituyen una guía que puede ser leída y comprendida por las personas destinadas a la producción del código, pruebas y mantenimiento del software.

Para el diseño de la aplicación, la metodología XP no requiere específicamente la presentación del sistema mediante diagramas UML, en su lugar, se usan otras técnicas como las tarjetas CRC y las metáforas (Fernández Escribano, 2002). No obstante, se emplearon los diagramas UML en el diseño de la solución debido a que influyen en el mejoramiento de la comunicación y no es costoso su mantenimiento en los casos aplicados. En general, el diseño debe proporcionar una idea completa del software enfocando los dominios de datos, funcional y de comportamiento, desde el punto de vista de la Implementación.

2.5.1 METÁFORA DEL SISTEMA

Una de las buenas prácticas de la metodología XP consiste en definir una metáfora para reflejar el funcionamiento del sistema, sirviendo de orientación en la fase de diseño. La metáfora es un medio de representar los objetos del dominio del sistema de modo que este pueda ser entendido por clientes y desarrolladores sin tener un conocimiento previo de dicho dominio (Beck, 2002), pues se compone de elementos presentes en la vida cotidiana, pero no debe interpretarse literalmente. Además, debe ejemplificar los componentes del sistema, ayudando a que todos tengan una idea del funcionamiento que se espera lograr en la aplicación a desarrollar, modelando una arquitectura lógica, no formal de dicha aplicación (Wake, 2002). De la misma manera facilita la retroalimentación entre el equipo completo en un lenguaje común accesible para todos.

Citando a (Beck, 2000): “La metáfora simplemente ayuda a todos en el proyecto a entender los elementos básicos y sus relaciones. Las palabras elegidas para identificar entidades técnicas deben estar reflejadas en la metáfora, la misma guiará el proceso de desarrollo, de modo que cada vez que se examine, el equipo de desarrollo podrá encontrar nuevas ideas”.

En aras de lograr una representación lógica de objetos del mundo real en el dominio de interés, y explicar los conceptos significativos del problema; se establece a continuación la metáfora del sistema.

- **Metáfora**: El sistema funciona como un archivador (organizador) de documentos donde al editar un documento se emplea un modelo con palabras predefinidas (editor), estas son combinadas para escribir en los documentos. Todo el sistema es utilizado por un administrador.

De manera descriptiva al explicar la metáfora, un administrador utiliza la aplicación a través de un *front-end* y este se vale del organizador para buscar y organizar los archivos de reglas. Para editar o modificar una regla la aplicación le ofrece al administrador un editor, el cual brinda una guía dinámica de palabras para hacer más fácil el proceso de construcción o edición de dicha regla.

2.5.1.1 Descripción de las entidades de la metáfora

Para lograr un mayor acercamiento al funcionamiento de los componentes del sistema, a continuación se describen las entidades que componen la metáfora antes descrita.

- **Administrador:** persona encargada de utilizar el sistema para la gestión de las reglas de detección. El usuario que usa el organizador permitiéndole visualizar los archivos, y emplea el editor para el trabajo con las reglas de detección.
- **Front-End:** es la “cara” del sistema para el administrador, debido a que constituye la interfaz con la que este interactúa para utilizar las funcionalidades que el sistema brinda.
- **Gestor de Reglas:** es un contenedor lógico para el organizador y el editor.
- **Organizador:** entidad que organiza los ficheros de reglas a través de un árbol, facilitando el trabajo con dichos ficheros y las reglas contenidas en estos.
- **Sensor:** ordenador en el cual se ejecuta Snort, y donde se encuentran los ficheros de reglas de detección.
- **Ficheros de Reglas:** entidad que contiene las reglas de detección.
- **Regla:** texto dentro de los ficheros de reglas que constituye las regla en sí.

2.5.1.2 Descripción de las relaciones entre las entidades de la metáfora

- **Administrador – Front-end:** el administrador maneja el sistema y sus funcionalidades a través del *front-end*, siendo este el medio que se emplea para interactuar con el sistema.
- **Front-End-Gestor de Reglas:** el *Front-end* se relaciona con dos entidades dentro del gestor de reglas: el organizador y el editor, mostrando la información enviada por dichas entidades y enviando la información en sentido inverso.
- **Organizador-Sensor:** el organizador busca los ficheros de reglas ubicados en un sensor manipulándolos para organizarlos conjuntamente con sus reglas.
- **Sensor-Fichero:** el sensor contiene uno o muchos ficheros de reglas en su sistema.
- **Fichero-Regla:** un fichero contiene una o muchas reglas.
- **Regla-Editor:** el editor principalmente modifica las reglas escritas en los archivos de reglas.
- **Organizador-Editor:** la información fluye de manera bidireccional entre las dos entidades para que estas puedan lograr su función en el sistema.

2.5.2 PATRONES ARQUITECTÓNICOS

A partir de la fuentes (Billy, 2013) y (Sandra Almeira, y otros, 2007) se sintetiza que la arquitectura de software es elemento fundamental del diseño de aplicaciones, tomando parte en la esquematización de los componentes de un sistema como entidades de un negocio determinado. El diseño arquitectónico debe permitir visualizar la interacción entre las entidades del sistema y describir en general cómo se construirá un sistema computacional. Es en este proceso donde toma lugar la aplicación de patrones de arquitectura.

Los patrones arquitectónicos son métodos de resolución de problemas en el campo de la ingeniería del software con gran aplicación en los modelos orientados a objetos (paradigma de orientación a objetos), sin embargo pueden ser utilizados en muchos ámbitos de la informática y las ciencias en general. La aplicación de patrones conlleva al establecimiento de una representación general de la organización y estructura de un sistema de software; definiendo un conjunto de elementos predefinidos, especificando responsabilidades, estableciendo reglas y guías para organizar las relaciones entre ellos (Sandra Almeira, y otros, 2007). Por esta razón, la correcta selección de los patrones a aplicar al dominio del presente problema de investigación tiene relevante importancia, produciendo un incremento positivo en la calidad del producto final.

En algunas bibliografías como (Urquiza Yllescas, y otros, 2010), se expresa que la mayoría de los sistemas de software no pueden ser estructurados con la utilización de un patrón arquitectónico simple, generalmente cuando se exige cubrir variadas piezas del dominio de un problema se requiere la aplicación de varios patrones arquitectónicos para estructurar tal sistema. Aunque los métodos de XP no exigen un establecimiento formal de una arquitectura, algunos autores (Auer, y otros, 2002) aprueban la aplicación de patrones arquitectónicos y de diseño, por ello y todo lo referido se precisa necesario la aplicación de estos en el diseño.

2.5.2.1 Modelo-Vista-Template

Como se explicaba en el epígrafe 1.5.2, el sistema se desarrolla bajo el marco de trabajo Django. Este marco implementa el patrón Modelo Vista Controlador (MVC) incorporando cierta variación en el concepto original. Los creadores de Django no buscaban ajustarse a la forma clásica del patrón cuando estaban inmersos en el proceso de construcción del marco de trabajo. Estaban interesados en encontrar una manera efectiva de desarrollar software pera la web, manteniendo los beneficios de MVC pero poniendo énfasis en facilitar el rápido desarrollo de una aplicación (Alchin, 2009). Como resultado establecieron una abstracción muy similar al modelo vista controlador llamada Modelo Vista Template (MVT) o Modelo Vista Plantilla.

Lo anterior se debe al funcionamiento interno del marco de trabajo. Este concibe parte de las funciones del patrón MVC clásico. El código se separa en capas lógicas: modelos, plantillas y vistas, de ahí el nombre del marco de trabajo. Esto viabiliza el mantenimiento y escalabilidad del sistema. A continuación se define el funcionamiento de las capas lógicas donde se muestra la similitud con MVC (Alchin, 2009):

- **Modelo**: permite tener separado de manera lógica los modelos de datos del resto de la aplicación. Uno de sus objetivos es mapear la base de datos creando una sincronización entre ese tipo de persistencia de datos y la aplicación, con el fin de mantener actualizada toda la información.
- **Vistas**: aunque comparte el mismo nombre que en MVC, las vistas en Django tienen una funcionalidad distinta. Estas combinan algunas responsabilidades de las vistas tradicionales y asumen totalmente la tarea de los controladores de MVC. De la misma forma, reciben la petición enviada por el navegador, procesan la información, realizan una petición al modelo para extraer información de la base de datos si es necesario y envía información de respuesta a la plantilla. Además, pueden ser

invocadas directamente como funciones, sin tener que definir una estructura aunque pueden establecerse como clases, funciones, funciones decoradas y objetos.

- **Plantillas**: la forma en que los datos son presentados está delegada a las plantillas (*templates*), de manera homóloga a la tarea de las vistas en el patrón MVC. Básicamente son archivos que contienen etiquetas HTML e instrucciones específicas del marco de trabajo, que tienen como objetivo mostrar en pantalla los contenidos de la forma deseada. Dichas instrucciones son simples y constituyen un lenguaje pequeño que Django proporciona, abstrayendo a los diseñadores de usar Python para el fin descrito.

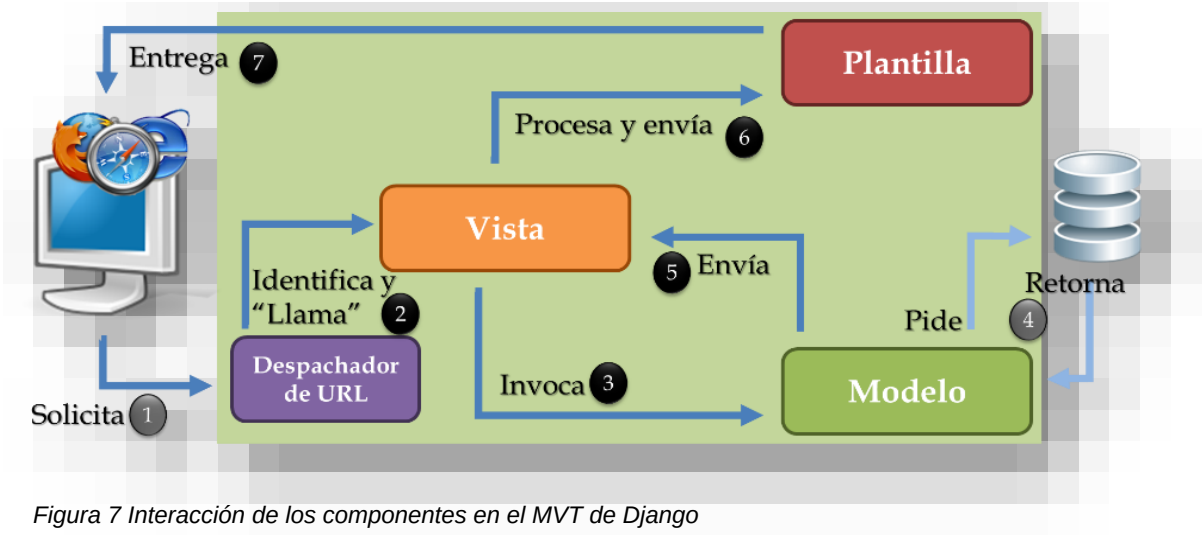


Figura 7 Interacción de los componentes en el MVT de Django

Existe un componente ligado a este marco de desarrollo de relevante importancia para su funcionamiento y es el despachador de URL (URL *Dispatcher*), el cual se encarga de recibir el requerimiento de una petición del cliente y enviarla a su respectiva vista para procesar la información. Generalmente se coloca en un archivo llamado urlconfig.py (Jordán Murillo, 2012). La Figura 7 muestra el funcionamiento de los componentes del MVT

2.5.2.2 Cliente-Servidor

Hoy día pocos sistemas de mediano y largo alcance se ejecutan en un solo ordenador, en su mayoría emplean las redes para realizar sus tareas. Una de las ventajas de los sistemas distribuidos es que permiten el compartimiento de recursos, la utilización de varios nodos de procesamiento. De la misma manera se distribuye el sistema de forma que este pueda realizar su función con sus componentes distribuidos o separados. Los sistemas distribuidos deben colaborar entre sí, y por ende necesitan comunicación entre ellos. Debido a que se quiere desarrollar un sistema que se pueda utilizar de manera remota y

funcionar con sus componentes ubicados en distintos lugares físicos; se consideró necesario la aplicación del modelo arquitectónico cliente-servidor.

De acuerdo con lo señalado en las fuentes (Luján Mora, 2002) y (Colorado Rodríguez, 2004), la arquitectura cliente-servidor es una arquitectura distribuida que separa una entidad “cliente” de otra u otras que actúan como “servidor”; denominando cliente aquel proceso que inicia el diálogo, solicita algún recurso o petición, y servidor al proceso que responde a las solicitudes del cliente a través de una interfaz de comunicación. Si bien existen muchos tipos de servidores, la arquitectura básica en estos no varía, permitiendo la construcción de aplicaciones distribuidas, teniendo como principal ventaja la posibilidad de separación de las funcionalidades de un sistema según su servicio, recursos o ambiente requerido para estas; permitiendo ubicar cada función en la plataforma más adecuada para su ejecución. Aunque el cliente y el servidor pueden actuar como entidades separadas, pueden residir en una sola entidad. Además, posibilita el acceso al sistema independientemente del lugar donde se encuentre el usuario, ya sea en plataformas separadas o en la misma plataforma del servidor.

2.5.3 DIAGRAMA LÓGICO DE LA ARQUITECTURA

Luego de haber establecido los patrones de arquitectura aplicados para la construcción de la aplicación; así como la metáfora que guiará el proceso de desarrollo, se procede a establecer un diagrama lógico de la arquitectura de sistema. Integrando en el mismo, tanto el marco de trabajo con MVT como la aplicación JS bajo la arquitectura cliente-servidor; y donde se contempla el resto del sistema del cual es parte el módulo de gestión de reglas en cuestión.

Algunas de las entidades que se muestran en Figura 8 como son el **front-end**, **fichero de reglas** y la **regla** ambos dentro del modelo, el **organizador**, el **editor** se derivan de la metáfora del sistema definida previamente.

El nodo **sistema del Proyecto IDS** representa la aplicación general de la cual el presente módulo es parte, esta recibe la entrada enviada por el *front-end* y se vale del **despachador de URL** para en caso que sea necesario, llamar al **Módulo de Gestión de Reglas de Detección,** objeto de esta investigación. Nótese que el **despachador de URL** es un componente del MVT de Django descrito en el epígrafe 2.5.2.1 previamente. Dentro del módulo, de acuerdo a la vista que el **despachador de URL** solicite, se activa una función determinada contenida en dicha vista. La vista que se denota como **manejador** se encarga

de gestionar cualquier llamada a otras vistas **auxiliares** si es necesario; luego invocan al modelo, donde se obtienen las reglas de detección de los ficheros dentro del equipo en el que **Snort** se ejecuta.

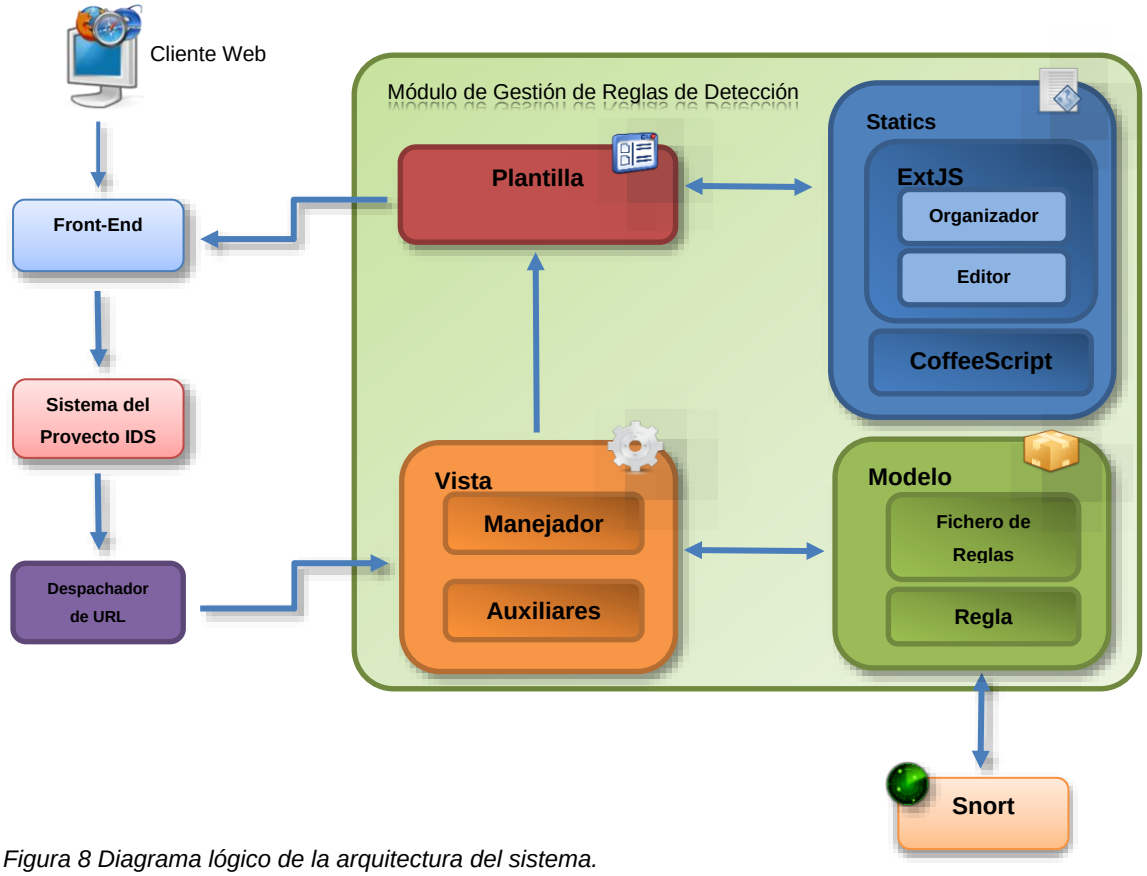


Figura 8 Diagrama lógico de la arquitectura del sistema.

La plantilla trabaja estrechamente con los componentes del ***statics***, lugar que emplea Django para almacenar varios recursos como las imágenes y los ficheros JS. Pues dentro del ***statics*** también están contenidos los ficheros de la implementación del **organizador** y el **editor** mediante la librería JavaScript **ExtJS** utilizada en conjunto con el lenguaje **CoffeScript**. Estos componentes viajan con la plantilla hacia el cliente para ser mostrados a través del *front-end*.

De esta manera quedan descritos todos los componentes de la arquitectura del módulo de gestión de reglas. Como se ha podido constatar, la aplicación de los patrones arquitectónicos está presente en el modelo arquitectónico descrito, integrando el MVT de Django con la arquitectura distribuida cliente-servidor. Es prudente mencionar que el **organizador** y el **editor** son aplicaciones JavaScript y por ende, se ejecutan completamente en el cliente. Esto responde directamente a la separación de funcionalidades del modelo distribuido escogido.

2.5.4 PATRONES DE DISEÑO

Los patrones de diseño son soluciones bien documentadas que los desarrolladores emplean para dar solución a problemas apoyados en la experiencia anterior de la resolución exitosa de los mismos problemas. Los profesionales identifican partes de un problema que son análogos a otros problemas que han resuelto anteriormente. Luego, retoman la solución utilizada y la generalizan. Por último, adecúan la solución general al contexto de su problema actual (Buschamann, y otros, 1996). Los patrones de diseño, en escala, son más pequeños en comparación con los patrones arquitectónicos. Lo que significa que están orientados a la resolución de problemas más específicos o de menor envergadura. Sin embargo se encuentran en un nivel superior a los lenguajes de programación, pues son independiente de estos.

Los patrones de diseño se han convertido en una técnica importante para la reutilización del conocimiento referente a la construcción de software (Sandra Almeira, y otros, 2007). Cada patrón provee información sobre el esquema de un sistema, describiendo las clases, métodos y relaciones que resuelven un problema de diseño en particular. Estos se agrupan en categorías que definen diferentes clasificaciones y descripciones.

2.5.4.1 GRASP asignación de responsabilidades

GRASP es un acrónimo que significa en inglés *General Responsibility Asignmemt Software Paterns* o patrones generales para la asignación de responsabilidades. Se escogió el nombre para indicar la importancia de captar (*grasping*) estos principios si se quiere diseñar eficazmente software orientado a objetos (Larman, 2004). A continuación se explica la aplicación de patrones pertenecientes al grupo de GRASP en el diseño y codificación del módulo de gestión de reglas.

El patrón experto consiste en asignar una funcionalidad al experto en la información relacionada con dicha funcionalidad. Es decir, la clase que cuenta con la información necesaria para cumplir con una responsabilidad determinada. De esta forma se garantiza que el conjunto de clases del sistema sea más fácil de entender, mantener y ampliar; presentándose la oportunidad de reutilizar los componentes en futuras aplicaciones (Larman, 2004). La aplicación de lo anterior permitió encapsular las funcionalidades o responsabilidades en las clases que contienen la información para cumplirlas, creando una

distribución del comportamiento entre las clases que cuentan con la información requerida para la realización de las operaciones del sistema.

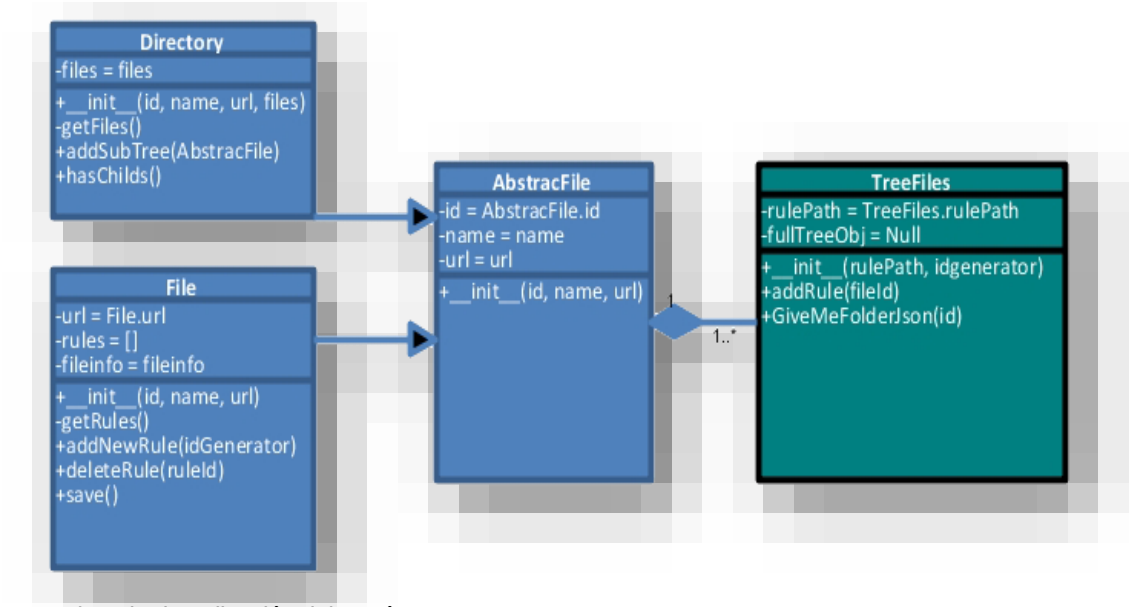


Figura 9 Ejemplo de aplicación del patrón experto.

En la Figura 10 se muestra un ejemplo de la aplicación del patrón experto, mostrando la asignación de la responsabilidad de adicionar una regla a la clase *TreeFiles*, experta en la información necesaria para ello. El cumplimiento de una responsabilidad requiere a menudo información distribuida en varios objetos, lo que significa que hay muchos expertos "parciales" que colaboran en una determinada funcionalidad.

En concordancia con la definición de la fuente (Larman, 2004), el patrón creador asigna la responsabilidad a una clase de crear una instancia de otra clase. El empleo de este patrón supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Con su empleo es probable que el acoplamiento no aumente, pues la clase creada tiene que ser visible a la clase creadora. Este patrón tuvo amplia aplicación en el módulo, presentándose en cada clase con la responsabilidad de crear otra u otras.

El patrón controlador concede la responsabilidad del manejo de mensajes de los eventos del sistema a una clase u objeto determinado, así como el envío de datos entre las unidades del sistema; definiendo un evento del sistema como un suceso de alto nivel generado por un actor externo o un evento de entrada externa; asociado generalmente a operaciones sobre la aplicación (Larman, 2004). Este puede ser visto una interfaz no destinada al usuario para el manejo de eventos. Su uso propone el diseño de clases con la responsabilidad de controlar el flujo de eventos del sistema.

La aplicación de este patrón GRASP aumenta la funcionalidad de los componentes o unidades reutilizables del sistema, garantizando que los procesos sean manejados por objetos separados de la interfaz gráfica de la aplicación. Igualmente contribuye a la reutilización de las operaciones del sistema en aplicaciones futuras. Todas las vistas del sistema son el equivalente a la aplicación del patrón controlador.

El patrón alta cohesión asocia responsabilidad de manera que exista una alta unión o cohesión. La cohesión está referida a cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas y que no realicen un trabajo excesivo (Larman, 2004). Tener en el código clases con una cohesión baja implica que estas sean difíciles de reutilizar y generalmente las afectan constantemente los cambios.

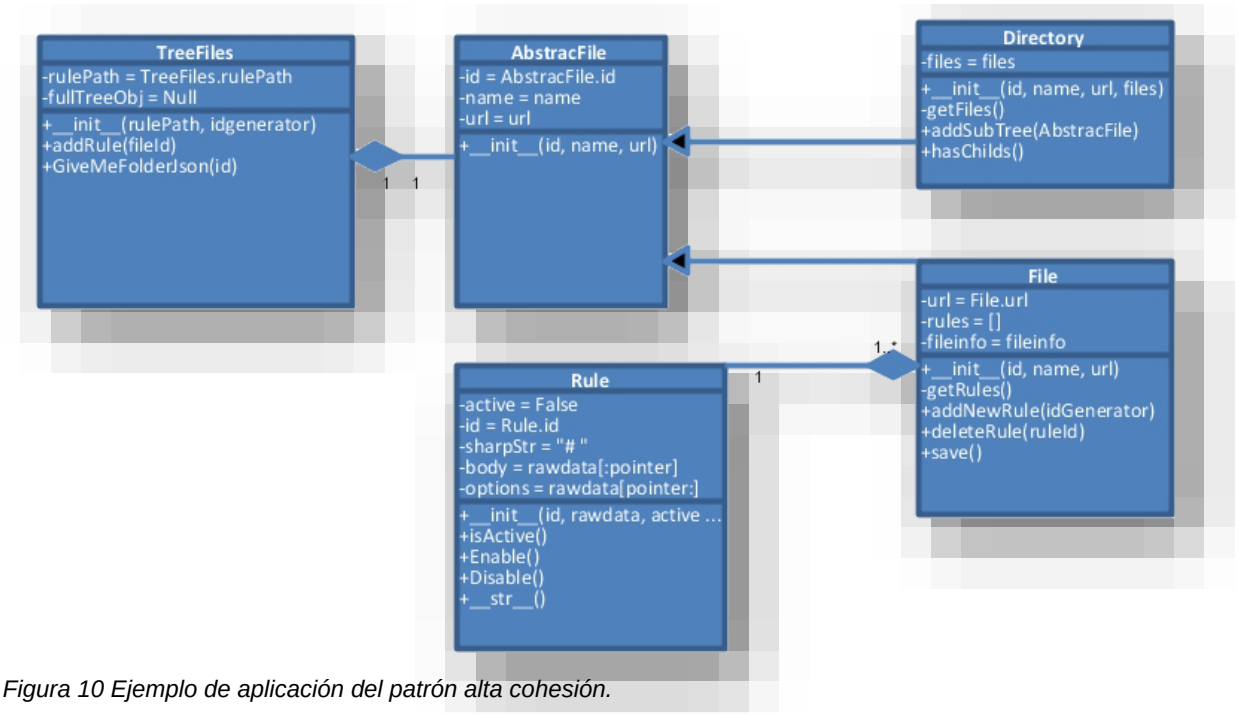


Figura 10 Ejemplo de aplicación del patrón alta cohesión.

La Figura 10 muestra un ejemplo de la alta cohesión de las clases y su estrecha relación que se manifiesta en las responsabilidades definidas, sin sobrecargar las tareas de ninguna de ellas. Es importante el uso de este patrón para mejorar la claridad y facilidad con la que se comprende el diseño y el código fuente. Además de que se simplifican las tareas de mantenimiento.

2.5.4.2 Patrones de comportamiento

Los patrones de comportamiento describen cómo organizar, administrar, y combinar conductas de objetos, centrándose en la comunicación entre ellos. Frecuentemente describen cómo los distintos elementos colaboran entre sí para conseguir un determinado objetivo. Pertenecen al grupo conocido como GOF o Banda de los Cuatro (del inglés *Gang of Four*) (Sandra Almeira, y otros, 2007).

El patrón Observador (*observer*) define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el elemento que se "observa" se encarga de notificar este cambio a todos los otros objetos dependientes (Freeman , y otros, 2004). En otras palabras, el objetivo principal es que un elemento observable notifique a sus observadores cada vez que se realiza un cambio en este. De acuerdo con el patrón se define que el elemento observable notifica a sus observadores y no los observadores los que constantemente comprueban si el observable ha sido modificado.

En la solución propuesta este patrón es aplicado al "observar" el estado de una regla de detección. Cuando esta se crea, modifica, o cambia su estado, la clase que representa dicha regla (que es el objeto "observado") notifica y envía la información resultante de las operaciones mencionadas al objeto que modela el árbol de ficheros, el cual se comporta como observador. De este modo se conoce en todo momento el estado de las reglas, sin tener que recurrir a la comprobación cada determinado tiempo.

2.5.4.3 Analizador Léxico y Analizador Sintáctico descendientes recursivos para gramáticas LL(1)

Para lograr la codificación inteligente que tributa al completamiento y resaltado de sintaxis del editor, se aplicaron los patrones analizador léxico (*scanner*) y analizador sintáctico (*parser*) del tipo descendente recursivo para gramáticas LL(1). De modo que la sintaxis para la construcción de reglas de detección se definió como un lenguaje y este pudo ser definido formalmente como una gramática para ser procesada por dichos patrones.

Los analizadores sintácticos descendentes recursivos construyen un árbol sintáctico de la raíz hacia las hojas, a partir del símbolo inicial de la gramática hacia los símbolos terminales; de aquí su descenso. Leen las entradas de izquierda a derecha (*left to right*) y va construyendo derivaciones o producciones de la gramática con el símbolo más a la

izquierda (*left most*). El número 1 de la denotación LL(1) indica el número de terminales que se consultan para decidir que regla de producción se aplica (Parr, 2009).

La tarea principal del analizador léxico consiste en devolver una secuencia de lexemas o *tokens*[30] a partir una lectura de caracteres que realiza como entrada, este también es llamado *tokenizer*. Este patrón puede reconocer estructuras léxicas anidadas (Parr, 2009). Además de su función principal, realiza otras secundarias como la eliminación de caracteres de tabulación, espacios en blanco, caracteres de fin de línea, entre otros que se pueden definir, aunque esto está determinado por la implementación que de analizador. En adición puede correlacionar notificaciones de errores con las sentencias que analiza.

Todo lenguaje tiene reglas que describen la estructura sintáctica de manera bien definida. Sin embargo es necesario determinar si ese lenguaje está escrito correctamente (Ravi, 1986). Para esto sirvió el Analizador sintáctico (*parser*) y es su objetivo principal, comprobar la correcta formación de las reglas de detección. Este patrón analiza la estructura sintáctica de la secuencia de *tokens* de un fragmento del lenguaje, comprobando que esta secuencia sea correcta en base a una gramática establecida. Otra de las funciones que realiza es el control de flujo de los *tokens* reconocidos y notificar la naturaleza de errores encontrados.

Las gramáticas constituyen la mejor vía para la descripción sintáctica de los lenguajes de programación ya que brindan una especificación precisa de los mismos (Ravi, 1986). A pesar de que no se describe un lenguaje de programación, sino un conjunto de sentencias que permiten crear las reglas que emplea Snort; se utilizó una gramática para definir la construcción de estas. La misma es del tipo libre de contexto según la jerarquía de Chomsky[31], por ello clasifica como gramática LL(1). Se empleó esta forma de representación debido a que una gramática bien diseñada ayuda al análisis de las sentencias del lenguaje y la detección de errores, además de que permite añadir nuevas construcciones al lenguaje de forma sencilla; por lo cual si se añaden opciones o sentencias en la construcción de las reglas en futuras versiones de Snort, el cambio en la actualización del módulo de gestión de reglas no será muy costoso.

[30] Indicio, señal o muestra de algún elemento previamente definido, por ejemplo un símbolo.
[31] Avram Noam Chomsky (Filadelfia, E.U., 7 /12/1928), Lingüista, filósofo y activista estadounidense. Profesor emérito de Lingüística en el MIT y una de las figuras más destacadas de la lingüística del siglo XX. A lo largo de su vida, ha ganado reconocimiento por su activismo político.

2.5.5 TARJETAS CRC

El uso de tarjetas Clase, Responsabilidad, Colaboración (del inglés *Class, Responsibilities, Collaboration*) es esencial en el diseño definido por XP. Permite involucrar a todo el equipo de desarrollo en la fase de diseño, nutriendo el proceso de muchas ideas y permitiendo tener un punto de vista más amplio en cuanto al modelado del sistema. Además logra romper con el modo secuencial de análisis y enfocar el proceso de construcción de una aplicación al paradigma de objetos (Donovan Wells, 1999). Específicamente, una tarjeta representa de manera muy sencilla objetos abstractos o concretos del dominio del sistema.

En cada tarjeta se escribe el nombre de la clase que representa y a la izquierda son definidas sus responsabilidades. En la derecha se establecen con qué clases colabora para dar cumplimiento a sus funciones. Dichas tarjetas deben escribirse de la manera más sencilla y transparente posible; haciendo que el diseño sea una tarea compartida por todo el equipo de desarrollo; de modo que se entienda el objetivo de cada componente o clase del sistema. No obstante, es posible agregar información a estas, como la clase de la que se especializa u otras que generaliza. De la misma manera es posible añadir los atributos de la clase o alguna otra descripción a la tarjeta CRC. A continuación se describen algunas de las tarjetas generadas en la fase de diseño, el resto se pueden ver en el anexo Tarjetas CRCTarjetas CRCTarjetas CRCTarjetas CRCTarjetas CRCTarjetas CRC.

Tabla 14 CRC clase Rule

Rule	
Modela una regla de un fichero de regla.	
Superclase	
Subclase	
Atributos	
Nombre	Descripción
id	Identificador de la regla
body	Cuerpo de la regla
options	Opciones de la regla
active	booleano true si está activa false si no
Responsabilidad	**Colaborador**
Saber si la regla está activa.	
Habilitar regla.	
Deshabilitar regla.	
Procesar la regla en bruto y colocar sus partes en los respectivos atributos de la clase.	

Devolver la regla.	
Devolver la regla en formato JSON.	

TreeFiles	
Modela una regla de un fichero de regla	
Superclase	
Subclase	
Atributos	
Nombre	Descripción
rulePath	URL de la regla
jsonCodec	Objeto para codificar datos a json
idGenerator	Objeto generador de identificadores
tree	Representación del árbol de directorios y ficheros.
jsonTree	Representación en json del objeto tree
Responsabilidad	**Colaborador**
Borrar regla	Files
Borrar fichero de regla	
Agregar regla	Files, JSONcodec, IDsClass
Deshabilitar una regla	Files
Habilitar una regla	Files
Dar los archivos de reglas en como objetos Json	Files, JSONcodec
Dar reglas de un archivo	Files, JSONcodec, IDsClass
Recargar árbol de directorios y ficheros	

Tabla 15 CRC clase TreeFiles

Capítulo 3 Construcción y Pruebas del Sistema

Lo elementos centrales de la implementación y materialización de la solución se ponen de manifiesto en el presente capítulo. Así como la descripción del proceso continuo de la realización de las pruebas a la aplicación. Ambos elementos realizados bajo las prácticas definidas por la metodología XP.

3.1 CONSTRUCCIÓN

La implementación de un sistema comienza con el resultado del diseño, donde en términos muy generales, responde a la relación existente entre las unidades que lo componen. El diseño se fomenta con la construcción de dichas unidades en términos de ficheros de código fuente, ficheros ejecutables, componentes entre otros (Melano, y otros, 2010). En XP el desarrollo es un proceso iterativo que incluye la constante refactorización del código, las pruebas unitarias (también conocidas como pruebas de unidad) y de aceptación.

El objetivo de la etapa de construcción es integrar y desarrollar por iteraciones todos los componentes, características y funcionalidades según las historias de usuario y el diseño de la aplicación. Al mismo tiempo que se desarrolla, se recomienda ir probando por partes, hasta obtener en la última iteración una versión aceptable del producto.

3.1.1 ESTÁNDARES DE CODIFICACIÓN

El estándar o estilo de codificación es el formato con que se escribe el código fuente en la implementación de una aplicación. El estándar varía en dependencia de lo acordado por el equipo de desarrollo y en ocasiones, del lenguaje de programación que se esté empleando. En un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los desarrolladores trabajen de forma homogénea.

La legibilidad del código fuente repercute directamente en la facilidad con que un programador comprende un sistema de software. El equipo de trabajo debe reflejar un estilo parejo, como si un único desarrollador hubiera escrito todo el código de una sola vez. Esto le otorga uniformidad a la escritura de la aplicación contribuyendo a todo mencionado anteriormente (MSDN, 2013). Es válido agregar que en un mismo proyecto se pueden establecer varios estándares de codificación según determine el equipo de trabajo.

El estándar de codificación que se adopta es el *UpperCamelCase* o en español Camello Elevado. Este define que los nombres de los diferentes elementos en la codificación están

formados por palabras que comienzan con mayúsculas seguidas por minúsculas. Igualmente, todos los nombres de las diferentes estructuras de código deberán ser descriptivos y toda la codificación debe ser escrita en idioma inglés.

3.1.1.1 Estructuras

Los nombres de las clases adoptarán el estilo definido sin emplear el guión bajo como delimitador entre palabras, lo cual se ilustra en la Figura 11.

```
1
2
3  class Academic_Course(object):          #incorrecto
4  class AcademicCourse(object):           #correcto
5
6
```

Figura 11 Estándar de codificación de clases.

La denotación para los atributos de las clases comenzarán con guión bajo seguidos por el nombre del atributo según la notación *lowerCamelCase* o en español Camello Bajo.

```
2
3  class AcademicCourse(object):
4          '''
5          Comment for this class
6          '''
7          def _init_(self, courseId):
8                  self._courseId = courseId       #correcto
9
```

Figura 12 Estándar de codificación para atributos.

El nombre de las propiedades y de las funciones se comportan de la misma manera que el nombre de las clases descritas anteriormente. En el caso del nombre de los parámetros pasados a funciones y variables dentro de las mismas utilizarán la variante *lowerCamelCase*; definiendo que todas las palabras que conformen el nombre a excepción de la primera comenzarán con la primera letra en mayúsculas y el resto en minúsculas. Todos estos nombres también deben ser descriptivos.

```
1
2   def AddTeacher(self, teacherId):              #correcto
3          newTeacher = "Teacher Information"     #correcto
4
5
```

Figura 13 Estándar de codificación para propiedades y funciones.

3.1.1.2 Documentación del código

Las clases con su documentación y sus propiedades deberán estar correctamente documentadas en forma de *docstring*[32] lo más descriptiva posible y en español; como se muestra en la Figura 14.

```
2
3  def AddTeacher(self, teacherId):
4          ´´´ Este método agrega un nuevo profesor a partir
5          del id del profesor´´´
6          if teacher > 0:
7
```

Figura 14 Estándar de codificación del docstring.

La documentación dentro del código se realizará en dependencia de la cantidad de líneas de código que se necesiten documentar. Para el caso de una sola línea, se escribe el signo de número antes del comentario. Para documentar varias líneas consecutivas se utilizará la forma *docstring*. Esta variante de comentario se empleará igualmente para el caso de una línea que su cantidad de caracteres impida leer el comentario al final de la misma. En la Figura 15 se muestran cada uno de los casos descritos.

```
2
3  def CreateCourse(self, courseId, currentSemester):
4          CourseId = courseId     #Comentario de linea
5          '''Comentario de bloque de código'''
6          if currentSemester > 2:
7                raise Exception("The currentSemester has to be one or two")
8          else:
9                CurrentSemester = currentSemester
10               return self
11
```

Figura 15 Estándar de codificación para comentarios.

3.1.1.3 Elementos XML

El estilo de notación que se adoptará para las etiquetas XML es el *UpperCamelCase*. De Igual forma los nombres deberán ser descriptivos. No obstante los comentarios en los documentos de extensión .xml se definirán por el estándar del lenguaje del mismo nombre que establece la sintaxis mostrada a en la Figura 16.

[32] Característica del lenguaje Python en español cadenas de documentación, la cual constituye una herramienta importante para documentar el código. Esta información es incluso accesible en tiempo de ejecución.

Todo el contenido explicado en este epígrafe se aplica a los componentes escritos en lenguaje *CoffeeScript* con extensión *.coffee*. Este lenguaje es utilizado en síntesis de *JavaScript* y este último es generado a partir del código *CoffeeScript*. Es decir, solo se dará soporte, revisión y se actualizarán los archivos *.coffee*. En consecuencia los estándares de codificación establecidos no se tendrán en cuenta para archivos *JavaScript*, debido a que se utiliza la codificación *CoffeeScript* en reemplazo a *JavaScript*.

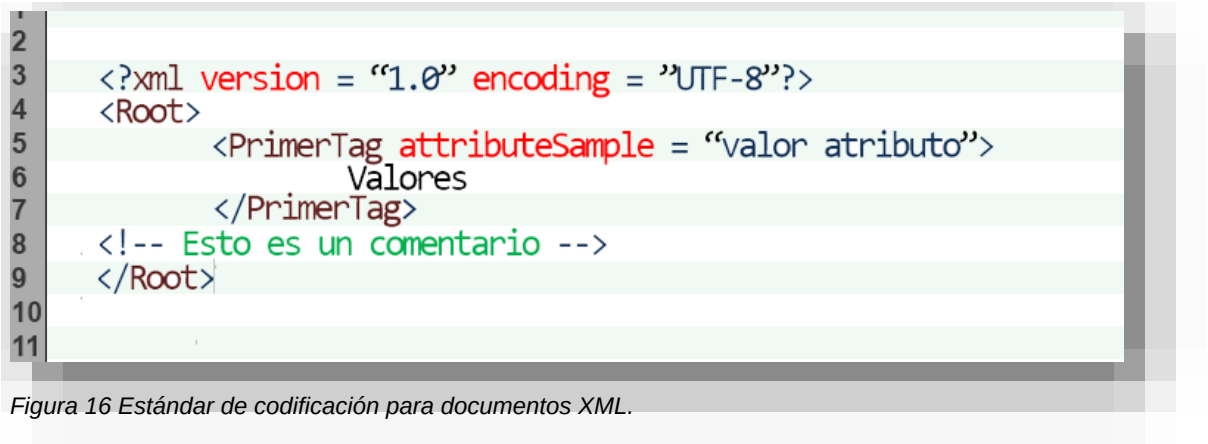


Figura 16 Estándar de codificación para documentos XML.

3.1.2 COMPONENTES DEL INTÉRPRETE PARA LA CODIFICACIÓN INTELIGENTE

En concordancia con los conceptos definidos en el epígrafe 2.5.4.3, se procede a la conformación de un autómata y una gramática, con el objetivo de implementar un intérprete que brinde asistencia al usuario en la construcción y edición de una regla de detección.

3.1.2.1 Definición de la gramática

Las gramáticas son esquemas generadores de un lenguaje determinado, las cuales establecen un modelo matemático que permite la especificación del mismo, determinado por un conjunto de reglas capaces de generar todas las posibles combinatorias de ese lenguaje y sólo las de dicho lenguaje, ya sea éste un lenguaje formal o un lenguaje natural (Aho, y otros, 2006). La sintaxis establecida por los desarrolladores de Snort para la construcción de las reglas, es representada por la gramática diseñada. Esto permite definir el lenguaje formal[33] empleado por el editor de reglas del sistema para reconocer las sintaxis de las reglas de detección. Teniendo como objetivo comprobar si una sentencia pertenece al lenguaje, o describir estructuradamente las sentencias que pertenecen al mismo.

[33] Son lenguajes más formalizados, como por ejemplo, los lenguajes de programación o el lenguaje matemático; tienen unas estructuras claramente definidas y determinadas por sus reglas gramaticales.

Como resultado se tiene que una gramática queda definida como la cuádrupla G = {N, ∑, P, S} donde:

- N es el conjunto de símbolos no terminales empleados para representar otros símbolos.
- ∑ es el conjunto arbitrario, finito y no vacío de símbolos que componen el alfabeto del lenguaje, llamados terminales.
- P es el conjunto de reglas de la gramática que permiten generar el lenguaje.
- S es el axioma de la gramática.

A continuación se exponen las reglas de producción iniciadas por el axioma.

```
          <regla> → <simple> <mas>
         <simple> → <cabecera>
            <mas> → <opciones> | ε
       <cabecera> → <accion> <protocolo> <origen> <sentido> <destino>
         <accion> → tk_alert | tk_log | tk_pass | tk_activate | tk_dynamic | tk_drop |
                    tk_reject | tk_sdrop
      <protocolo> → tk_tcp | tk_icmp | tk_udp | tk_ip
         <origen> → <direccion> <Puerto>
      <direccion> → tk_ipd | tk_cidr | tk_httpserver | tk_externlanet
         <puerto> → tk_any | tk_entero | tk_http_ports | <listaPuerto>
    <listaPuerto> → tk_coropen tk_ entero <otroPuerto> tk_corclose
     <otroPuerto> → tk_coma tk_entero <otroPuerto> | ε
        <sentido> → tk_forward |tk_bothsides
        <destino> → <direccion> <Puerto>
       <opciones> → tk_paren_open <listaOpciones> tk_paren_close
  <listaOpciones> → <unaOpcion> tk_ptcoma <masOpciones>
      <unaOpcion> → <opcion>
    <masOpciones> → <listaOpciones> | ε
         <opcion> → <msg> | <ref> | <gid> | <sid> | <rev> | <class> | <priority> | <meta>
            <msg> → tk_msg tk_dospt tk_cadena
            <ref> → tk_reference tk_dospt <idURL> tk_coma tk_id
          <idURL> → tk_bugtraq | tk_cve | tk_nessus | tk_arachnids | tk_mcafee | tk_osvdb |
                    tk_url
            <gid> → tk_gid tk_dospt tk_entero
            <sid> → tk_sid tk_dospt tk_entero
            <rev> → tk_rev tk_dospt tk_entero
          <class> → tk_classtype tk_dospt <classname>
      <classname> → tk_attempted_admin | tk_attempted_user | tk_inappropriate-content |
                    tk_policy_violation | tk_shellcode_detect | tk_successful_admin |
                    tk_successful_user | tk_trojan-activity | tk_unsuccessful_user |
                    tk_web_application_attack | tk_attempted_dos | tk_attempted_recon |
                    tk_bad_unknown | tk_default_login_attempt | tk_denial_of_service |
                    tk_misc_attack | tk_non_standard_protocol | tk_rpc_portmap_decode |
                    tk_successful_dos | tk_successful_recon_largescale |
                    tk_successful_recon_limited | tk_suspicious_filename_detect |
                    tk_suspicious_login | tk_system_call_detect |
                    tk_unusual_client_port_connection | tk_web_application_activity |
                    tk_icmp_event | tk_misc_activity | tk_network_scan | tk_not_suspicious |
```

```
                tk_protocol_command_decode | tk_string_detect | tk_unknown |
                tk_tcp_connection
    <priority> → tk_priority tk_dospt tk_entero
        <meta> → tk_metadata tk_dospt <cuerpoMeta>
  <cuerpoMeta> → <unCuerpoMeta> <masCuerpoMeta>
<unCuerpoMeta> → tk_engine_shared | tk_soid tk_entero tk_or tk_entero | tk_service_http
<masCuerpoMeta> → tk_coma <unCuerpoMeta> | ε
```

De acuerdo con el alcance de la presente investigación, la gramática comprueba que la regla cumpla con su estructura básica, las sintaxis y semántica de las opciones generales según el manual de usuario de Snort (Roesch, y otros, 2012) en su versión 2.9.3.

3.1.2.2 Autómata finito determinista

Los autómatas finitos (AF) como esquemas reconocedores dentro de los mecanismos de representación de lenguajes, poseen gran utilidad para la implementación de un analizador léxico; siendo este último, necesario para la construcción del editor con codificación inteligente. Un autómata finito o máquina de estado finito, es una herramienta abstracta que se utiliza para reconocer un determinado lenguaje regular (Aho, y otros, 2006). Es un modelo matemático de un sistema que recibe una cadena constituida por caracteres de cierto alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

Para la utilización de un AF en la presente investigación, se hace necesario que este sea del tipo determinista. Un autómata finito determinista (AFD) es aquel cuyo estado de llegada está unívocamente determinado por el estado de salida y el carácter leído por el autómata. (Aho, y otros, 2006). Es decir, para cada carácter reconocido, solo habrá un siguiente estado posible al cual se puede llegar.

Para la representación del AFD se empelará un diagrama de Moore o de transición como también es conocido, mostrado en la Figura 17. En este, los estados son definidos por círculos rotulados con el nombre del estado. El estado inicial se encierra dentro de un cuadrado y los estados finales o terminales son los círculos con fondo verde. Además, las transiciones se representan con arcos entre dos estados etiquetados con el símbolo por el cual se pasa al estado siguiente. El diagrama de transición se empleó específicamente en el reconocimiento de los *tokens* del lenguaje de las reglas de detección. Los asteriscos significan que en el correspondiente estado no se consumirá el último carácter leído.

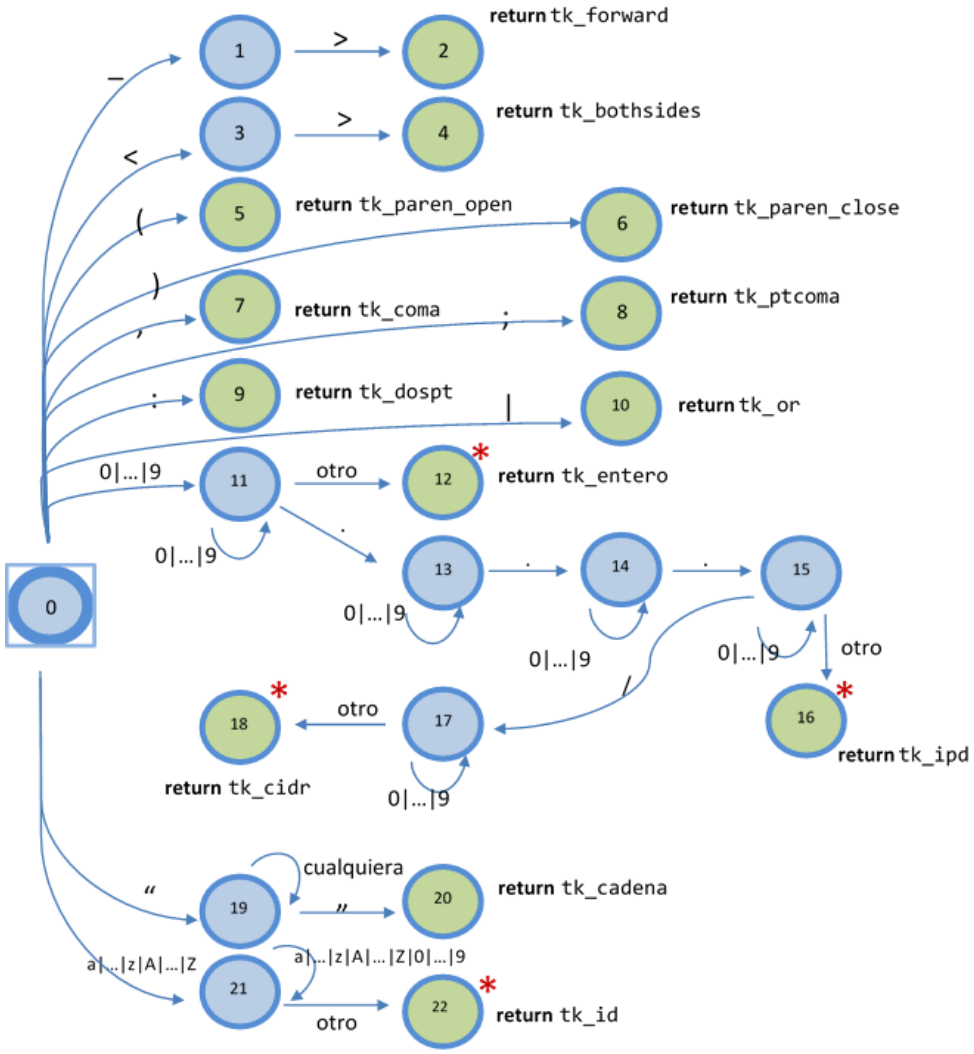


Figura 17 Diagrama de transición del autómata de la gramática.

3.1.3 DESCRIPCIÓN DE LA INTERFAZ GRÁFICA DE USUARIO

El prototipo de interfaz gráfica de usuario se diseñó en concordancia con las propiedades del sistema que el cliente definió en las historias de usuario correspondientes. Es una interfaz sencilla, orientada a íconos y no sobrecargada de funcionalidades. A pesar de ser una aplicación web, su GUI está ambientada en la apariencia de una aplicación de escritorio.

Todas las funcionalidades del sistema están agrupadas en secciones y son de fácil acceso. En la sección izquierda se localiza el árbol de directorios, ficheros y reglas sirviendo de explorador general, brindando un rápido acceso a cualquier elemento a procesar. Sobre los ficheros de reglas se activa un menú contextual dando acceso a las funcionalidades adicionar y eliminar regla del fichero. De manera adicional, muestra una información visual

acerca de cuál regla se encuentra activa en el fichero y cuál desactivada. La Figura 18 constituye una vista del prototipo de interfaz de usuario.

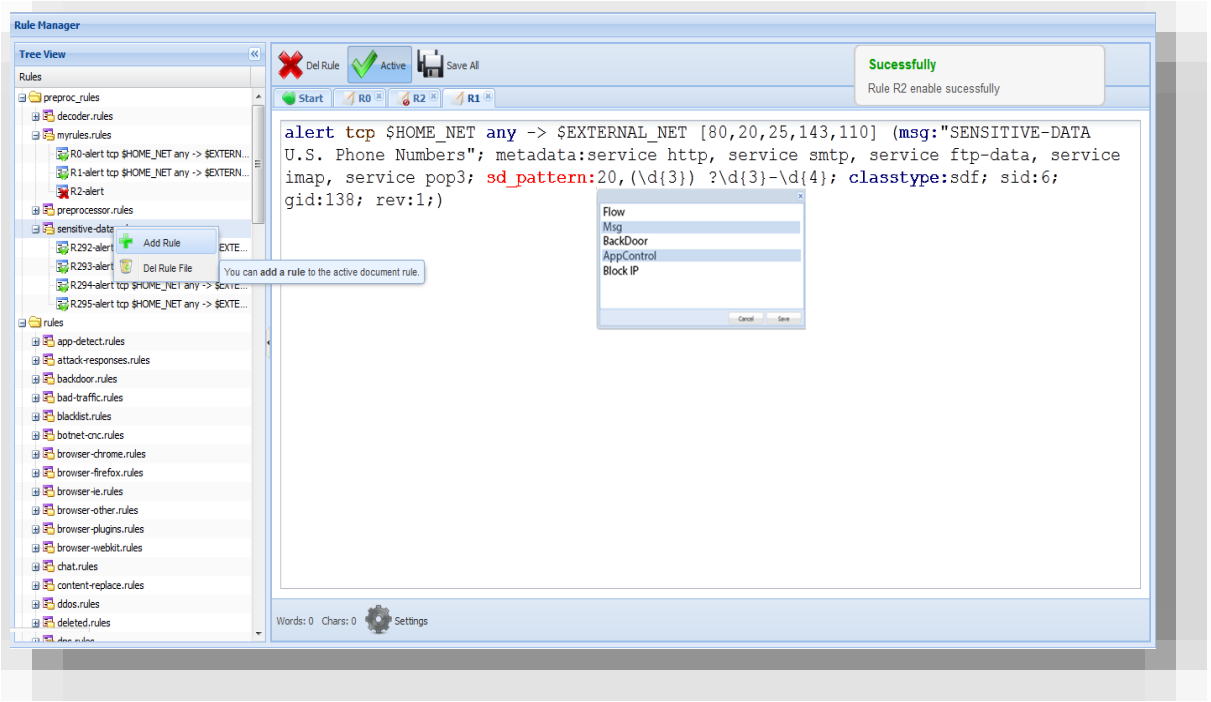


Figura 18 Prototipo de la interfaz principal de la aplicación.

La zona de la derecha siendo la más amplia se encuentra dividida en tres secciones. En el nivel superior se encuentra una barra de tareas, conteniendo funcionalidades del sistema, permitiendo deshabilitar una regla o activarla, suprimirla del fichero o salvar todas las reglas abiertas pertenecientes a un fichero. El área central se compone por la barra de etiquetas, donde se añade una de estas cada vez que el usuario decide editar una nueva regla. En el área de texto se lleva a cabo el proceso de codificación inteligente donde el usuario es guiado por la aplicación al construir o modificar una regla. En el tratamiento del código contenido en el área de texto, la aplicación muestra palabra reservadas, valores esperados con completamiento y señalamiento de sintaxis incorrecta.

En el último nivel de la zona central se encuentra la barra de estado. Esta provee información al usuario del proceso relacionado con la edición de la regla de detección. Se encuentra además un botón que proporciona el paso a una ventana de configuración donde se establece la ubicación del sensor para ubicar el archivo de configuración de Snort. De esta manera la aplicación obtiene la localización de los ficheros de reglas para construir el árbol.

Cabe agregar que la aplicación muestra luego de cada acción una notificación en la parte superior derecha; brindando retroalimentación al usuario con respecto a lo sucedido después de que una acción fue llevada a cabo. En otras palabras, muestra errores o información relacionada con el resultado luego de ejecutar una funcionalidad. De igual manera se define que todos los elementos cuentan con una ayuda rápida de cada funcionalidad presente en la interfaz. Esta ayuda se muestra a manera de contenedor en forma de globo con información respecto a la acción señalada.

3.1.4 DIAGRAMA DE DESPLIEGUE

El diagrama o modelo de despliegue como también es conocido, muestra la distribución de los nodos de procesamiento en tiempo de ejecución, los enlaces de comunicación entre estos y las instancias de componentes y objetos que residen en estos. Su objetivo es capturar la configuración entre los elementos (*hardware* o *software*) en el sistema (Rational Software Corporation , 2003). Los nodos que componen el modelo simbolizan componentes que contienen al menos un elemento de procesamiento, memoria u otro dispositivo en su interior.

Estos nodos también pueden representar dispositivos, los cuales constituyen nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción modelado. Los nodos son relacionados a través conectores y pueden representar procesos, permitiendo reflejar la distribución del comportamiento entre los nodos. XP no define formalmente la generación de un modelo de despliegue de manera explícita. Sin embargo se cree pertinente la definición del diagrama para reflejar la disposición de los componentes del sistema. El modelo se muestra en la Figura 19.

3.1.4.1 Descripción de los nodos

Ordenador cliente: Este nodo aloja el navegador web a través del cual el usuario accede al *front-end* de la aplicación y las funcionalidades de este como el organizador de archivos y las funcionalidades de codificación.

Servidor de Aplicación: En este nodo se ejecuta el módulo Gestor de Reglas de Snort sobre apache v2.2.21 o superior. Es importante mencionar que en este nodo debe ejecutar la aplicación Apache HTTP Server v2.2.x o superior con el módulo wsgi v3.3. Este módulo se encarga de atribuir a Apache la capacidad de servir aplicaciones escritas en Python; vale mencionar que existen diversas maneras de lograr esto.

Sensor: Este nodo representa el ordenador donde Snort se ejecuta y del cual se procesarán las reglas de detección.

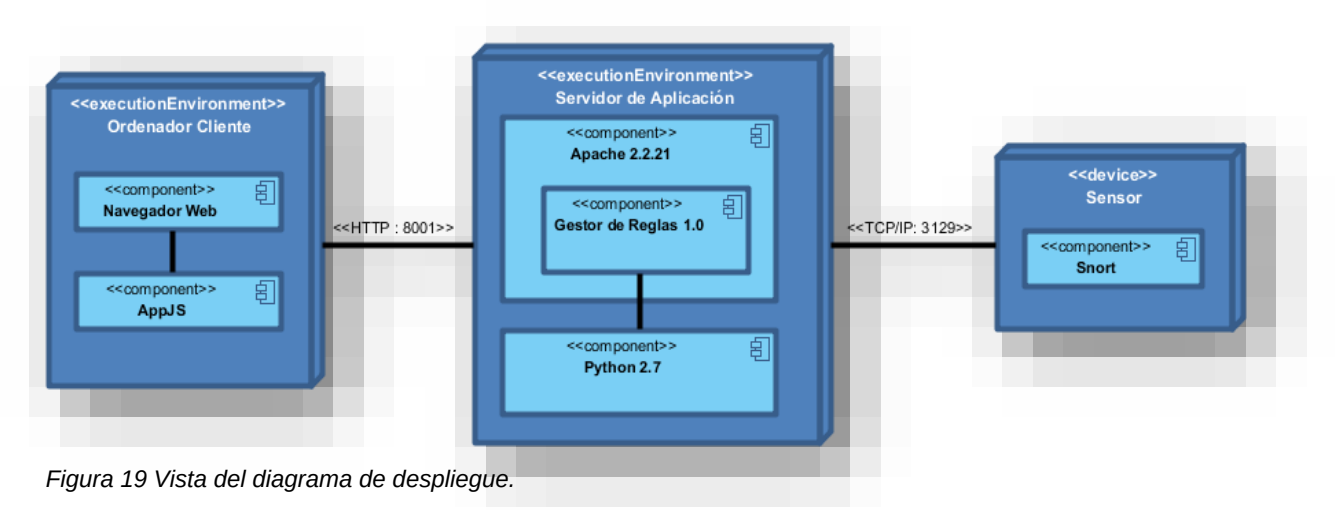


Figura 19 Vista del diagrama de despliegue.

Debido a la interacción del usuario con el sistema mediante el *front-end* utilizando un navegador web, el nodo ordenador cliente siempre iniciará la comunicación de peticiones a través del protocolo HTTP por el puerto 8001. Luego, en dependencia de la solicitud recibida, el sistema utiliza el protocolo TCP\IP para acceder a las reglas de detección alojadas en el sensor. La información fluye en modo inverso al descrito en las situaciones que se envían datos al ordenador cliente.

3.2 PRUEBAS

La calidad es un factor importante en el proceso de desarrollo de software y el instrumento justo para garantizar dicha calidad son las pruebas de software. Una prueba de software es aquella actividad encaminada a evaluar las capacidades de un programa o sistema, constituyendo una manera eficaz de comprobar si cumple con los resultados esperados (Hetzel, 1992).

En XP las pruebas son un elemento base para el desarrollo de la calidad del software. No obstante esta metodología define la estrategia de pruebas un tanto distintivo de otras. En primera instancia, establece que las pruebas no son una fase, sino un proceso continuo y repetitivo que toma lugar probando e integrando pequeñas unidades del software; tratando de mitigar los problemas a los que se enfrentan los probadores al ejecutar pruebas de sistema, en aplicaciones que nunca han sido probadas a nivel de unidad o de integración (Crispin, y otros, 2002). Lo anterior ocasiona que en este punto se pueda solucionar muy

poco para ganar en calidad del producto final, pues solo se puede detectar de lo que el sistema carece o lo que presenta.

De acuerdo con lo establecido por XP, se recomienda escribir durante todo el desarrollo del software dos tipos de pruebas: pruebas unitarias y pruebas de aceptación (Beck, 2000). Las primeras son escritas por los desarrolladores y las segundas por el cliente. Los programadores escriben pruebas unitarias de manera que el código se comporte como ellos esperan; y el cliente escribe pruebas de aceptación basándose en lo que desean que la aplicación realice. A medida que el proceso se repite por iteración, el resultado final es un programa con la calidad esperada y más adaptable al cambio. Adicionalmente, la programación en parejas hace que detectar y corregir errores en el código durante la codificación del mismo, sea factible.

3.2.1 PRUEBAS UNITARIAS

Las pruebas unitarias están dirigidas a probar clases y métodos, lo que las hace estar relacionadas intrínsecamente con el código y la responsabilidad de cada clase y sus secciones más críticas. El objetivo es comprobar que la clase, como unidad funcional de un programa, este correctamente codificada (Juristo, y otros, 2005). La realización de este tipo de pruebas ayuda a asegurar la calidad del código auxiliando la detección de errores en situaciones tempranas de desarrollo. De igual forma las pruebas de unidad constituyen una guía para el programador, permitiéndole obtener una medida de cómo está realizando su trabajo.

De acuerdo con las prácticas de XP se definen las pruebas unitarias completamente automatizadas. Estas son escritas antes de escribir el código de una clase o unidad. Cuando es concluida la codificación de una unidad, se ejecuta la prueba sobre la misma, probando sus métodos y responsabilidades. Esto significa que no se usan métodos tradicionales para realizar las pruebas unitarias, por lo que estas serán llevadas a cabo por los programadores. Definir las pruebas antes de programar hace más sencillo y rápido codificar ya que la prueba en sí funciona como guía para lo que se quiere lograr.

Las pruebas automatizadas son sencillas rutinas que controlan el funcionamiento del código. En alternativa a entrar datos manualmente al sistema en aras de comprobar como este se comporta, o emplear un método convencional como las pruebas de caja blanca; las pruebas automatizadas se realizan durante todo el proceso de construcción de manera automática y desatendida a cierto nivel. Estas son ejecutadas por el mismo sistema (Django

Software Foundation , 2013). Una vez que se escribe un conjunto de pruebas, a medida que se realizan cambios en la codificación de la aplicación se puede chequear si el código trabaja como se espera. Esto sin tener que empelar tiempo realizando pruebas manuales.

3.2.1.1 Módulo UnitTest2 de Django

Probar una aplicación web en Django puede ser una tarea compleja, debido a que se compone de varias capas lógicas: desde el protocolo HTTP al nivel de la administración de solicitudes, luego la validación de los formularios y el procesamiento, hasta la representación de la plantilla (Django Software Foundation, 2013). Con el *test-execution*[34] *framework* y el módulo *UnitTest2* ambos pertenecientes a Django, se pueden simular peticiones y respuestas, introducir datos de prueba, inspeccionar la salida de la aplicación y verificar que el código está haciendo lo que debería hacer.

```
from django.utils import unittest
from radmin.model.files import TreeFiles
from radmin.model.tools.Tools import IDsClass

class TreeFielsTC(unittest.TestCase):
    def setUp(self):
        self.idGenerator = IDsClass()
        self.tree = TreeFiles('C:\Users\Link\Desktop\snortrules', self.idGenerator)

    def TestAddRule(self):
        """
        Testin addRule method
        """
        result = self.tree.addRule('F1')
        self.assertIn('"success"', result, 'AddRule works.')

    def TestDelRule(self):
        """
        Testing delRule method
        """
        result = self.tree.delRule('R0', 'F0')
        self.assertTrue(result, 'TestDelRule Works.')

    def TestDelRuleFile(self):
        """
        Testign DelRuleFile method
        """
        self.assertTrue(self.tree.delRuleFile('F0'), 'DelRuleFile works.')

    def TestDisableRule(self):
        """
        Testing DisableRule method
        """
        self.assertTrue(self.tree.disableRule('R0', 'F0'), 'DisableRule works.')
```

Figura 20 Fragmento de código del caso de prueba TreeFilesTC.

El módulo UnitTest2 de Django está escrito a partir de la librería *Unittest* de Python v2.7; con el objetivo de poder aprovechar las bondades de la librería, reescrita en la versión de

[34] *Test-Execution Framwork,* en español marco de trabajo prueba-ejecución es un componente interno usado por Django para el testeo de aplicaciones.

Python mencionada. Las pruebas unitarias se llevaron a cabo de manera automatizada, con el empleo del módulo antes indicado.

El caso de prueba (del inglés *test case*) *TreeFilesTC* mostrado en la Figura 20, constituye la prueba unitaria automatizada para la clase TreeFile, de la cual previamente se definió su responsabilidad y colaboradores en las tarjetas CRC del epígrafe 2.5.5. Esta pertenece a la capa del modelo y se encarga más bien de funciones del proceso de negocio. Luego de estar confeccionado, se pueden realizar las pruebas de unidad para la clase y sus colaboradores, confirmando de esta manera si el funcionamiento de la misma es el deseado.

Luego de correr la prueba se muestra información sobre la ejecución y el resultado de la misma. Si una prueba del CP falle se muestra una notificación del error, como la mostrada en la Figura 22; donde la imagen muestra dos pruebas fallidas para el presente CP, de un total de 5 realizadas. La Figura 21 muestra la misma prueba con resultados satisfactorios.

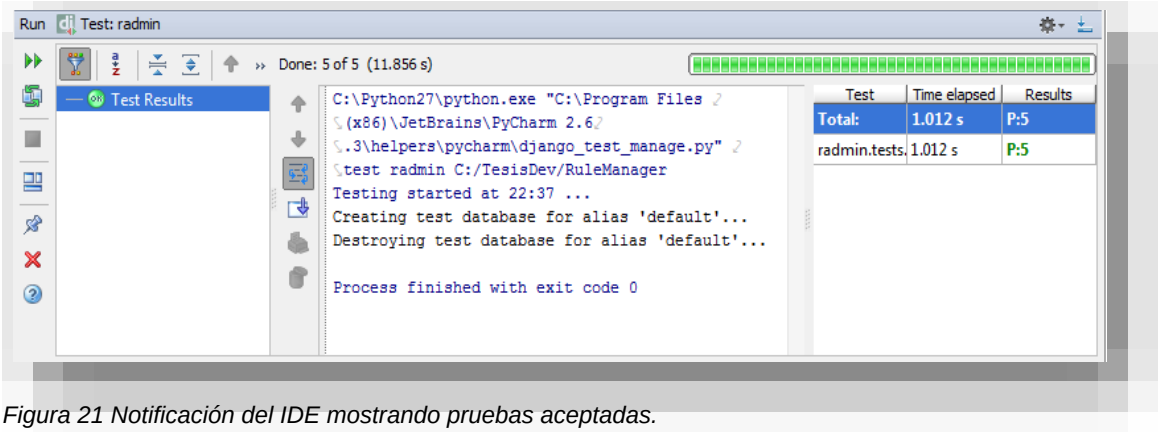


Figura 21 Notificación del IDE mostrando pruebas aceptadas.

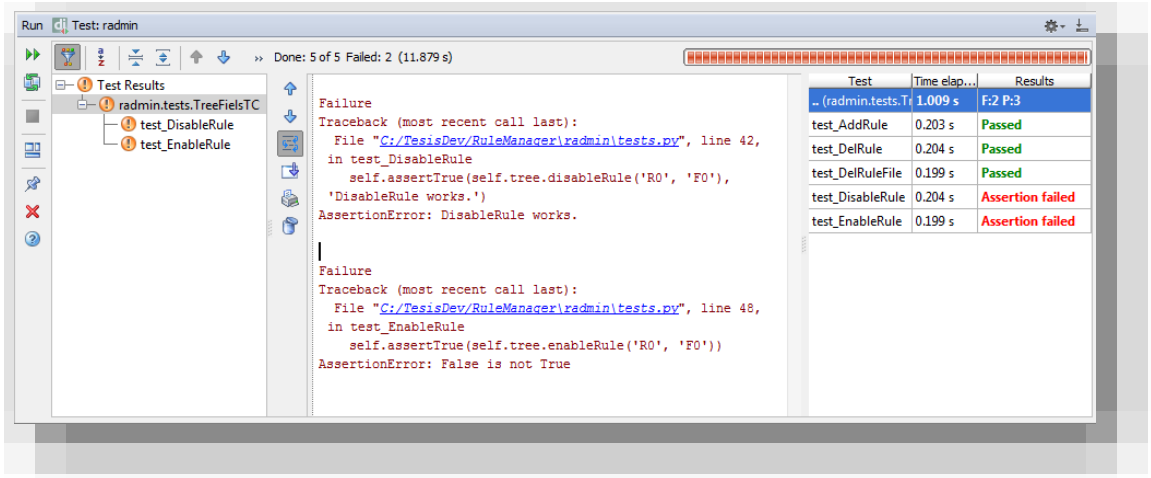


Figura 22 Notificación del IDE mostrando pruebas fallidas.

3.2.1.2 Pruebas unitaria para la vista principal vmain

Debido a que el presente sistema es una aplicación web, probar unidades correspondientes a las distintas capas hace variar en cierta medida a las pruebas unitarias. Por ello se hace uso de una clase *client* que modela un navegador web enviando peticiones para probar la respuesta del sistema. Permitiendo interactuar con todos los niveles de la aplicación. Debido a que simula peticiones POST y GET; así como la supervisión de las vistas que son llamadas para determinada URL. Además que posibilita comprobar el renderizado de las plantillas de Django y sus variables (Django Software Foundation, 2013). En la Figura 23 se muestra un segmento del CP *VmainTC* correspondiente a la vista principal del sistema.

En el fragmento del CP presentado en la Figura 23, el código visualizado comprueba que se muestre la página principal de la aplicación, si el servidor responde la cifra 200, significa que se ha retornado un "*ok*", expresando que el proceso se realizó correctamente. El otro método inspecciona si se devuelve una respuesta satisfactoria a la petición de leer árbol de ficheros, pasando como parámetro el id de un directorio. Es válido esclarecer que en el código de los CP, específicamente en el nombre de las funciones, no se puede cumplir cabalmente el estándar de codificación definido previamente, por el hecho de que el módulo *UnitTest* obliga a poner la palabra *test* con guión bajo (*test_*) antes del nombre de la función; artificio que utiliza internamente para reconocer cada prueba dentro de un CP.

3.2.2 PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación son un elemento integral del desarrollo guiado por XP, y se encuentran en un nivel superior a las pruebas unitarias. Cada historia de usuario da paso a una o varias pruebas de aceptación. Estas son definidas totalmente por el cliente y representan la revisión de sus requerimientos, el cual debe confeccionar sus pruebas teniendo en cuenta lo que se precisa verificar, para establecer que la historia de usuario asociada a la prueba está terminada en su totalidad (Beck, 2000). No obstante, el equipo de desarrollo necesita supervisar las pruebas de aceptación.

En este proceso se necesita completa retroalimentación entre los desarrolladores y el cliente, dado a que este último en la mayoría de los casos conoce qué procedimientos desea que la aplicación realice correctamente, pero puede no ser capaz de desarrollar un conjunto de pruebas que garanticen la total cobertura de la funcionalidad especificada en la historia de usuario (Crispin, y otros, 2002). Limitándose a comprobar que el sistema se comporta como debe sin verificar todas las variantes que pueden aparecer. Algunas

bibliografías definen las pruebas de aceptación como pruebas de caja negra (Myers, 2004) (Malfará, y otros, 2006). Estas pruebas ayudan a refinar las funcionalidades en cada iteración marcando el camino a seguir por el equipo de desarrollo y luego de ejecutarlas, se esclarece qué falta por hacer.

```
from django.test import TestCase
from radmin.model.files import TreeFiles
from radmin.model.tools.Tools import IDsClass

class VmainTC(TestCase):
    def setUp(self):

        self.idGenerator = IDsClass()
        self.tree = TreeFiles('C:\Users\Link\Desktop\snortrules', self.idGenerator)

    def test_Home(self):
        response = self.client.get('')
        self.assertEqual(response.status_code, 200)

    def test_ReadTree(self):
        kwargs = {'HTTP_X_REQUESTED_WITH': 'XMLHttpRequest'}
        response = self.client.get('/readtree/', {'id': 'R1'})
        self.assertIn('sucess:true', response.context, kwargs)
```

Figura 23 Fragmento de código del caso de prueba VmainTC.

La definición de los casos de prueba confeccionados por el cliente, describen la ejecución de la prueba de aceptación de una historia de usuario. Además, establecen las condiciones en las que deben ser revisados, la iteración en que son realizados, la criticidad de la funcionalidad, así como la acción que se supervisará. Las tablas a continuación muestran el resumen de los CP definidos para las HU1 y HU2 y HU4 en la iteración número 3.

Construir árbol de ficheros y reglas		**HU1**
Iteración	3	
Crítico	Si	
Función	Se debe mostrar el árbol de directorios, ficheros y reglas.	
Precondiciones	Se debe haber introducido el camino al fichero de configuración de Snort.	
Acción	**Datos**	**Resultado Esperado**
Entrar comino	URL válido	Cargar el árbol
Entrar camino	URL incorrecto	Notificar error

Tabla 16 CP historia de usuario 1

Crear Regla	**HU2**
Iteración	3

Crítico	No	
Función	Debe agregar una regla nueva al fichero.	
Precondiciones	Tener un fichero de reglas seleccionado en el árbol.	
Acción	**Datos**	**Resultado Esperado**
Hacer clic derecho		Es mostrado el menú contextual
Hacer clic en la acción “añadir regla”	Fichero seleccionado	Muestra ventana de confirmación
Marcar “si”	Cadena “Si” Id del fichero seleccionado	Mostrar regla indexada en el fichero seleccionado y confirmación de acción satisfactoria
Marcar “no”	Cadena “No”	Nada es mostrado

Tabla 17 CP historia de usuario 2

Eliminar Regla		**HU4**
Iteración	3	
Crítico	Si	
Función	Eliminar la regla del fichero al cual pertenece.	
Precondiciones	Tener una regla activa en el editor del sistema.	
Acción	**Datos**	**Resultado Esperado**
Hacer clic en el botón “Borrar regla”	Id del botón Id de la regla Id del fichero	Es mostrada una notificación como resultado del proceso.

Tabla 18 CP historia de usuario 4

En el proceso de construcción de la aplicación, los CP correspondientes a las funcionalidades se emplean solo como guía para la realización de las pruebas. Para automatizar y documentar el proceso se recurrió a la herramienta Selenium.

Selenium dota a los navegadores de una automatización orientada principalmente a pruebas de caja negra. Se empleó Selenium IDE en su versión 2.0; instalándose como complemento en el navegador Mozilla Firefox para su utilización, pues permite realizar comprobaciones del contenido de una página cargada, supervisar un campo con determinado id y su valor, entre otras funciones (Selenium Project, 2012). Igualmente, es posible convertir la automatización de las pruebas a varios lenguajes para ser usados como *test* unitarios, uno de estos lenguajes es Python.

La herramienta se centra en grabar todas las operaciones que se realizan sobre una aplicación web registrando las acciones, los datos que son enviados y recibidos, así como los resultados de las acciones realizadas, es decir el comportamiento de la aplicación en general. De este modo Selenium permite comprobar y documentar de manera ágil los casos

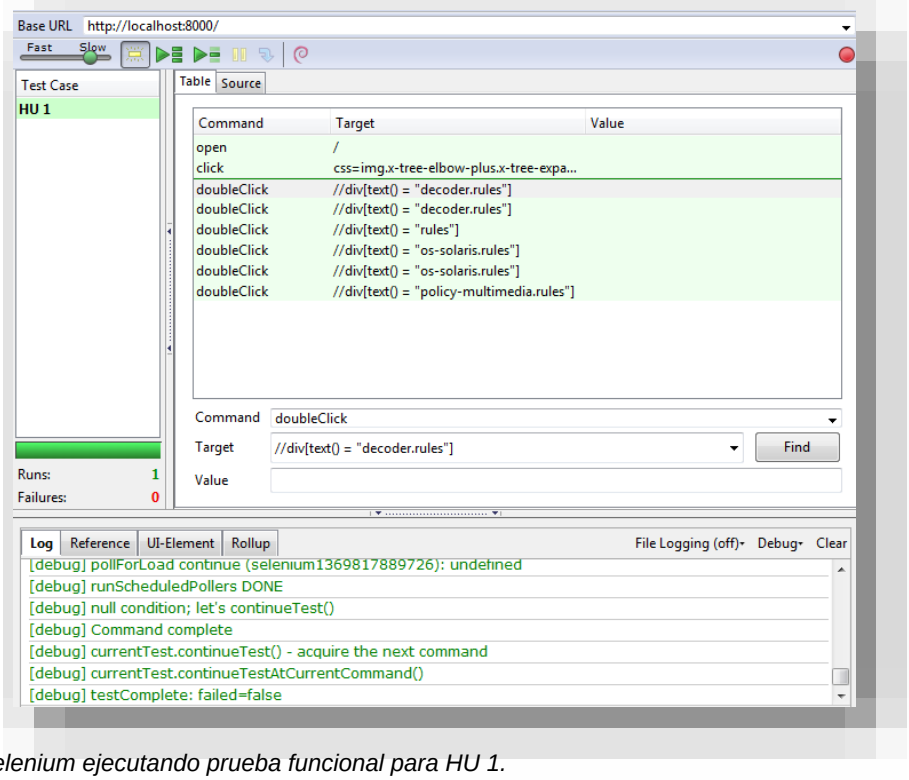
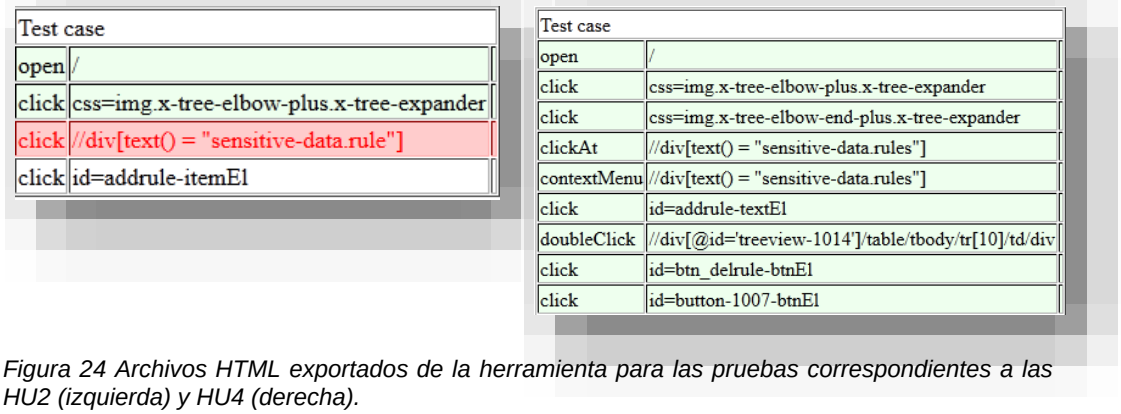


Figura 25 Selenium ejecutando prueba funcional para HU 1.

de prueba definidos por el cliente. Cabe mencionar que la herramienta permite la administración de las pruebas de manera persistente, posibilitando gestionar CP de varios sistemas para su posterior reutilización o modificación, incluso dichos casos pueden pertenecer a aplicaciones distintas.



Test case	
open	/
click	css=img.x-tree-elbow-plus.x-tree-expander
click	//div[text() = "sensitive-data.rule"]
click	id=addrule-itemEl

Test case	
open	/
click	css=img.x-tree-elbow-plus.x-tree-expander
click	css=img.x-tree-elbow-end-plus.x-tree-expander
clickAt	//div[text() = "sensitive-data.rules"]
contextMenu	//div[text() = "sensitive-data.rules"]
click	id=addrule-textEl
doubleClick	//div[@id='treeview-1014']/table/tbody/tr[10]/td/div
click	id=btn_delrule-btnEl
click	id=button-1007-btnEl

Figura 24 Archivos HTML exportados de la herramienta para las pruebas correspondientes a las HU2 (izquierda) y HU4 (derecha).

En la Figura 25 se puede observar la ejecución de la prueba de aceptación automatizada para la HU1 en la tercera iteración de pruebas. La imagen muestra las acciones realizadas sobre componentes de la página, así como las respuestas que se obtienen visualizadas en la parte inferior de la ventana. La herramienta se puede configurar para que genere como salida un archivo HTML con el resultado de la prueba, las operaciones realizadas y valores de parámetros pasados al servidor. En la Figura 24 se muestra este tipo de salida para las HU2 y HU4 respectivamente.

La información contenida en el archivo HTML exportado es algo pobre en contenido, por ello lo recomendable es administrar toda la información de las pruebas directamente en Selenium IDE. Esto permite obtener una vista completa del diseño y la información de los CP. Por consiguiente, el proceso de pruebas de aceptación fluye de manera eficiente, asi como su documentación y su supervición.

3.2.3 RESULTADOS

El proceso de desarrollo del sistema fue guiado totalmente por las pruebas: unitarias en un nivel inferior y pruebas de aceptación en un nivel superior. Si todas las pruebas en una iteración fueron satisfactorias, se procedió pasar a la siguiente. Además, los casos de prueba exitosos en iteraciones anteriores fueron comprobados también en la siguiente.

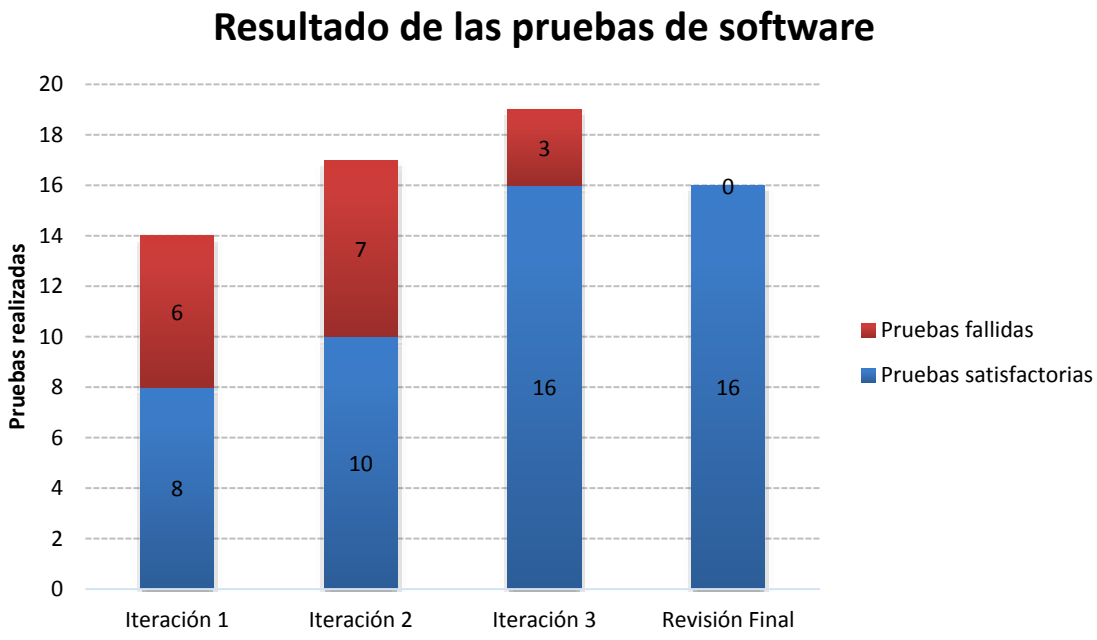


Figura 26 Prueba unitarias y de aceptación que se dieron como aceptadas y fallidas en las iteraciones del ciclo de desarrollo del producto.

Al finalizar la última iteración definida en la planificación, se realizaron 17 pruebas de aceptación y 18 pruebas unitarias, todas satisfactorias en la última etapa. Comprobando así que la aplicación cumple con las historias definidas por el cliente. La Figura 26 muestra la relación de las pruebas realizadas por iteración, correlacionadas con las pruebas fallidas de la iteración. El número total de *test* está definido adicionando pruebas de unidad y de aceptación. Se puede apreciar que en la revisión final, luego de la tercera iteración, todas las pruebas fueron satisfactorias.

Conclusiones

Luego de las consideraciones anteriores se arriban a las siguientes conclusiones:

- La síntesis realizada a partir del estudio de las tendencias actuales del proceso de gestión de las reglas de detección de Snort, permitieron adquirir una mejor visión y entendimiento de este proceso; además de poder identificar características claves a incorporar en el sistema desarrollado.
- Siendo Snort un IDS que basa su detección en reglas, estas son un factor crítico y determinante en la calidad de su funcionamiento, por tanto resulta necesario que el proceso de gestión de dichas reglas sea asistido para tratar evitar errores que comprometan la eficacia del IDS.
- Las tecnologías libres escogidas en el desarrollo web tales como ExtJS, Django y Python garantizaron en un entorno de calidad, la construcción de un sistema funcional independiente de la plataforma en que se utilice, cumpliendo con los objetivos trazados en el inicio del trabajo.
- La aplicación de patrones arquitectónicos en el diseño, particularmente los patrones analizador léxico y sintáctico; posibilitaron la implementación del editor de reglas, que brinda asistencia al usuario permitiendo una codificación inteligente, sin tener que acceder físicamente al fichero que contiene a dichas reglas.
- Con la metodología XP guiando el ciclo de vida del sistema y la aplicación de sus buenas prácticas garantizaron cumplir con la calidad requerida en el producto final.
- El proceso de desarrollo guiado por pruebas, como lo define XP, permitió validar funcionalmente el sistema desarrollado; comenzando por la realización de pruebas de unidad en fases tempranas, y pruebas de aceptación, llevadas a cabo por el cliente para certificar el completamiento de la aplicación.

Recomendaciones

Como recomendaciones de la presente investigación, se tienen:

1. Añadir al reconocimiento del editor, las opciones de reglas pertenecientes a otras clasificaciones diferentes de la implementada.
2. Probar la aplicación desarrollada en las redes de la Universidad.

Bibliografía Referenciada

Rational Software Corporation . 2003. *Rational Unified Process [Full Help v.2003.06.00.65].* s.l. : Rational Software Corporation., 2003.

Aho, Alfred V., et al. 2006. *Compilers: Principles, Techniques, and Tools [Second Edition].* Stanford : Pearson Education Inc., 2006. 0-321-48681-1.

Alchin, Marty. 2009. *Pro Django.* New York : Apress, 2009. 978-1-4302-1048-1.

Alfons. 2009. Seguridad y Redes. *IDS Policy Manager v3. Configuración sensores snort remotos.* [Online] 2009. [Cited: febrero 18, 2013.] http://seguridadyredes.wordpress.com/2010/02/10/ids-policy-manager-v3-configuracion-sensores-snort-remotos-actualizacion-desde-v22/.

Alfons. 2010. Seguridad y Redes. *Snorby un front-end para análisis y gestión de alertas para Snort.* [Online] 2010. [Cited: febrero 18, 2013.] http://seguridadyredes.wordpress.com/2010/12/01/snort-snorby-un-front-end-para-analisis-y-gestian-de-alertas-para-snort/.

Allan, Ant. 2005. *Intrusion Detection Systems (IDSs): Perspective.* s.l. : Gartner, 2005. DPRO-95367.

Álvarez, Miguel Angel. 2001. Desarrolloweb.com. *Una introducción meramente conceptual al potente lenguaje de script del lado del cliente.* [Online] 2001. [Cited: febrero 8, 2013.] http://www.desarrolloweb.com/articulos/25.php.

América-Economía. 2012. América-Economía. *Estudio: delitos informáticos se expanden a redes sociales y a móviles.* [Online] 2012. [Cited: febrero 17, 2013.] http://tecno.americaeconomia.com/noticias/estudio-delitos-informaticos-se-expanden-redes-sociales-y-moviles.

Auer, K and Miller, R. 2002. *Extreme Programming Applied.* s.l. : Addison-Wesley, 2002.

Bace, Rebecca Gurley. 2000. *Intrusion Detection.* Indianapolis : Macmillan Technical Publishing, 2000. 1-57870-185-6.

Bajo de Luque, Marías Jesús . 2011. Metodologías Ágiles. [Online] 2011. [Cited: marzo 6, 2013.] http://metodologiasagiles.herobo.com/index.php/es/2011-12-05-16-09-55/metodologia-xp/planificacion.

Beck, Kent and Flower, Martin. 2000. *Planing Extreme Programming.* New Jersey : Addison-Wesley, 2000. 0-201-71091-9.

Beck, Kent. 2000. *Extreme Programming Explained.* New Jersey : Addison-Wesley, 2000. 201-61641-6.

Beck, Kent. 2002. *The Metaphor Metaphor.* Los Angeles : Presentation at OOPSLA, 2002.

Billy, Carlos Reynoso. 2013. carlosreynoso. [Online] 2013. [Cited: marzo 8, 2013.] http://carlosreynoso.com.ar/archivos/arquitectura/Introduccion.PDF.

Blogsfarm Network SL . 2011. ProgramacionDesarrollo.es. *blog sobre programación y desarrollo web.* [Online] 2011. [Cited: febrero 11, 2013.] http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/.

Bradenbaugh, Jerry. 2000. *Aplicaciones JavaScript.* Madrid : Anaya Multimedia, 2000. 84-415--1070-9.

Britos, José Daniel. 2010. *Detección de Intrusiones en redes de datos con captura distribuida y procesamiento estadístico.* Madrid : [Tesis de Maestría], 2010.

Buschamann, Frank, et al. 1996. *Pattern - Oriented Software Architecture. A system of Patterns.* England : Jhon Wiley & Sons, 1996. 0-471-95869-7.

Cockbun, A. and Williams, L. 2000. *The Costs and Benefits of Pair Programming.* s.l. : Humans and Technology Technical Report, 2000.

Cohn, Mike. 2012. Succeeding with Agile - Mike Cohn's Blog. *Non-functional Requirements as User Stories.* [Online] 2012. [Cited: marzo 5, 2013.] http://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories#comments. 1-888-61.

Colorado Rodríguez, César. 2004. *Diseño y desarrollo de aplicaciones web multidispositivo.* s.l. : Germinus XXI, 2004.

Community of Snorby. 2013. Snorby. *All About Simplicity.* [Online] 2013. [Cited: febrero 18, 2013.] https://snorby.org/ .

Contrera Chávez, Nataly. 2011. Scribd. [Online] Scribd Inc., 2011. [Cited: enero 31, 2013.] http://es.scribd.com/doc/59484913/Historia-de-La-Seguridad-a.

Crispin, Lisa and House, Tip. 2002. *Testing Extreme Programming.* Illinois : Addison Wesley, 2002. 0-321-11355-1.

De Luca, Damián. 2010. CSS3 & HTML5. *¿Qué es CSS3?* [Online] 2010. [Cited: febrero 9, 2013.] http://css3html5.com.ar/que-es-css3/.

Django Software Foundation . 2013. Django. *Writing your first Django app, part 5.* [Online] Django Software Foundation , 2013. [Cited: mayo 24, 2013.] https://docs.djangoproject.com/en/1.5/intro/tutorial05/#writing-your-first-django-app-part-5.

Django Software Foundation. 2013. Django. *Testing Django applications.* [Online] Django Software Foundation, 2013. [Cited: mayo 24, 2013.] https://docs.djangoproject.com/en/1.4/topics/testing/.

Django Software Fundation. 2013. Django. [Online] Django Software Fundation, 2013. [Cited: febrero 9, 2013.] https://www.djangoproject.com/.

Donovan Wells, J. 1999. XPrograming.com. [Online] 1999. [Cited: mayo 20, 2013.] http://www.xprogramming.com/Practices/PracCRC.html.

ECMA Script Group. 2013. Introducing JSON. [Online] ECMA Script, 2013. [Cited: marzo 22, 2013.] http://www.json.org/.

Erlijman Piwen, Ariel and Goyém Fros, Alejandro. 2001. *Problemas y Soluciones en la Implementación de Extreme Programing.* Montevideo : Universide de Montevideo, 2001.

esCERT. 2009. esCERT UPC. [Online] 2009. [Cited: febrero 2, 2013.] https://escert.upc.edu/content/curso-de-concienciaci%C3%B3n-medidas-b%C3%A1sicas-de-seguridad.

Fernández Escribano, Gerardo. 2002. [Online] 2002. [Cited: marzo 8, 2013.] http://www.clubdevelopers.com/prog/articulos/xp/downloads/xp.pdf.

Fiebel, Werner. 2005. *The Encyclopedia of Networking [second edition].* EEUU : Sybex, 2005. 0-7821-1829-1.

Freeman , Elisabeth and Freeman, Eric. 2004. *Head First Design Patterns.* s.l. : O'Relly, 2004.

Fronckowiak, John W. 2008. *Build Ajax applications with Ext JS.* s.l. : IBM Corporations., 2008.

García Alfaro, Juaquín . 2005. *Detección de Ataques de Red con Snort.* Valencia : A5, 2005.

Gonzales Duque, Raúl. 2010. *Python para todos.* España : s.n., 2010.

González Gómez, Diego. 2003. *Sistemas de Detección de Intrusiones.* 2003.

Hetzel, William C. 1992. *The Complete Guide to Software Testing.* Canada : John Wiley and Sons, 1992. 0471565679.

Holovaty, Adrian and Kaplan-Moss, Jacob. 2009. *The Definitive Guide to Django [Second Edition].* New York : Apress, 2009. 978-1-4302-1937-8.

Jeffries , Ron, Anderson , Ann and Hendrickson , Chet. 2001. *Extreme Programming Installed.* New jersey : Addison-Wesley, 2001. 0-201-70842-6.

JetBrains. 2013. JetBrains.com. *World's leading vendor of Smart Developer Tools.* [Online] 2013. [Cited: febrero 11, 2013.] http://www.jetbrains.com/pycharm/features/.

Jordán Murillo, Carlos Luis . 2012. Tuneando la WEB. [Online] 2012. [Cited: febrero 19, 2013.] http://tuneandolaweb.blogspot.com/2012/04/descripcion-de-un-modelo-template-view.html.

Joskowicz, José. 2008. *Reglas y Prácticas en eXtreme Programming.* España : Doctorado de Ingeniería Telemática de la Universidad de Vigo, 2008.

Juristo, Natalia, Moreno, Ana M. and Vegas, Sira . 2005. *Técnicas de Evaluacón de Software.* 2005.

Larman, Craig. 2004. *UML y Patrones. Intoducción al análisis y diseño orientado a objetos.* La Habana : Félix Varela, 2004.

Letelier, Patricio and Penadés, María del Carmen. 2009. *Métodologías ágiles para el desarrollo de software: Extreme Programing (XP).* Valencia : Universidad Politécnica de Valencia, 2009.

López, Marisa. 2011. Scribd. [Online] 2011. [Cited: febrero 18, 2013.] http://es.scribd.com/doc/125519155/Pfc-Marisa-Capitulo3.

Lucena López, Manuel J. 2005. *Criptogrfía y seguridad en Computadores [4ta Edición].* España : s.n., 2005.

Luján Mora, Sergio. 2002. *Programación de aplicaciones web: Historia, principios básicos y clientes web.* s.l. : Editorial Club Universitario, 2002. 9788484542063.

MacCaw, Alex. 2012. *The Little Book on CoffeeScript.* USA : O'Reilly, 2012. 978-1-449-32105-5.

Malfará, Delvis and Cukerman, Diego. 2006. *Testing en Extreme programing.* Madrid : s.n., 2006.

Mansfield, Richard. 2005. *CSS Web Design for Dummies.* Indianapolis : Wiley Publishing Inc., 2005. 0-7645-8425-1.

McRee, Russ. 2007. *Activeworx IDS Policy Manager 2.0: Rules management for multiple sensors.* s.l. : ISSA Journal, 2007.

Melano, Santiago, Molina , Cecilia and German, Paneli. 2010. *Sistema para la Administración de Programas y Projectos.* Río Cuarto : Univesisdad Nacional de Río Cuarto [Tesis], 2010.

Mendoza Sanchez, Maria A. 2004. Metodologías de Desarrollo de Software. *Informatízate.* [Online] 2004. [Cited: febrero 6, 2013.] http://www.informatizate.net/metod.

MSDN. 2013. [Online] Microsoft Corporation, 2013. [Cited: mayo 20, 2013.] http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

Myers, Glenford J. 2004. *The Art of Sotware Testing [second edition].* s.l. : Jhon Wiley & Sons inc., 2004.

Nguyen, Chi. 2007. Mosaic. [Online] 2007. [Cited: febrero 18, 2013.] http://mosaic.cnfolio.com/M591CW2007B104.

Oinkmaster. 2013. Oinkmaster. [Online] 2013. [Cited: febrero 18, 2013.] http://oinkmaster.sourceforge.net/features.shtml.

OMG. 1997. OMG we set the standard. [Online] OMG, 1997. [Cited: febrero 7, 2013.] http://www.omg.org/gettingstarted/what_is_uml.htm.

OSSIM. 2013. SourceForge. *OSSIM, The Open Source SIEM.* [Online] SourceForge by Dice Holdings, Inc Company, 2013. [Cited: febrero 18, 2013.] http://sourceforge.net/mail/?group_id=86016.

Parr, Terence. 2009. *Language Implementation Patterns. Create you Own Domain-Specific and General Programming Languages.* Dallas : The Pragmatic Bookshelf, 2009. 978-1-934356-45-6.

Párrigas, Angel Alonso. 2010. *Osim, una plataforma clave para la seguridad en profundidad.* Madrid : hakin9.org, 2010.

Pino, Carlos. 2009. Administración de Sitemas Operativos. [Online] 2009. [Cited: febrero 4, 2013.] http://www.adminso.es/index.php/Archivo:Pfc_Carlos_cap3.pdf.

Ponemon Institute. 2011. *Second Annual Cost of Cyber Crime Study [Benchmark Study of U.S. Companies].* EEUU : Ponemon Institute LLC. [Sponsored by ArcSight, an HP Company], 2011.

PR Newswire Association LLC. 2013. PR Newswire. *a UBM plc company.* [Online] 2013. [Cited: febrero 11, 2013.] http://www.prnewswire.com/news-releases/con-pycharm-los-desarrolladores-de-python-finalmente-obtienen-una-ide-poderosa-104948549.html.

Pressman, Roger S. 2010. *Software Engineering A Practitioner´s Approach [seventh edition].* New York : McGrawHill Higher Education, 2010. 978-0-07-337597-7.

Puchades Olmos, Adrian. 2008. *Análisis de la Plataforma OSSIM. Sistema de Gestión de Información.* Valencia : Universidad Politécnica de Valencia, 2008.

Pulledpork. 2013. Pulledpork for Snort rule management. [Online] 2013. [Cited: febrero 18, 2013.] https://code.google.com/p/pulledpork/.

Python Software Fundation. 2013. Python. [Online] Python Software Fundation, 2013. [Cited: febrero 9, 2013.] http://www.python.org/about/.

Raggett, Dave. 2003. W3C. [Online] 2003. [Cited: febrero 8, 2013.] http://www.w3.org/MarkUp/Guide/.

Ramió Aguirre, Jorge. 2004. *Seguridad Informática y Criptografía.* Madrid : Universidad Politécnica de Madrid, 2004. 84-86451-69-8.

Ravi, Sauman. 1986. *Compilers: principles, techniques, and tools.* Boston, USA : Addison Wesley .Inc, 1986.

Real Academia Española. 2012. Diccionario de la Lengua Española [Vigésima segunda edición]. [Online] Real Academia Española, 2012. [Cited: 6 16, 2013.] http://lema.rae.es/drae/?val=seguridad.

Rehman, Rafeeq Ur. 2003. *Intrusion Detection System with Snort.* New Jersey : Prentice Hall PTR, 2003. 0-13-140733-3.

Roesch, Martin and Green, Chris. 2012. *Snort User Manual 2.9.3.* s.l. : Sourcefire inc., 2012.

Sandra Almeira, Adriana and Perez Cavenago, Vanina. 2007. *Arquitectura de Software: Estilos y Patrones [Tesis].* San Juan Bosco, Argentina : Facultad de Ingeniería de la Universidad Nacional de la Patagonia, 2007.

Scott, Charlie , Wolfe, Paul and Hayes, Bert. 2004. *Snort for Dummies.* Indianapolis : Wiley Publishing, Inc, 2004. 0-7645-6835-3.

Selenium Project. 2012. Selenium HQ Browse Automation. [Online] Selenium Project, 2012. [Cited: mayo 29, 2013.] http://docs.seleniumhq.org/docs/.

Shea, Frederick, et al. 2010. *Learning Ext JS 3.2.* Birmingham UK : Packt Publishing, 2010. 978-1-849511-20-9.

Sourcefire Inc. 2010. Snort. *Snort FAQs.* [Online] 2010. [Cited: febrero 3, 2013.] http://www.snort.org/snort/faq/#2.2.

Sourcefire. 2013. Snort. [Online] Sourcefire, 2013. [Cited: 01 11, 2013.] http://www.snort.org/snort.

StaridsLabs. 2012. StaridsLabs. *Intrusion Detection System.* [Online] mayo 2012. [Cited: febrero 18, 2013.] http://staridlabs.org/notes/IDS.html.

Tangient, (Wiki). 2013. Programación-Extrema Wiki. [Online] 2013. [Cited: marzo 2013, 4.] http://programacion-extrema.wikispaces.com/4.+Artefactos.

Tori, Carlos. 2008. *Hacking Ético.* Rosario, Argentina : s.n., 2008. 978-987-05-4364-0.

Urquiza Yllescas, José Fidel, Martínez M, Alfonso and Ibargüengoitia G, Guadalupe. 2010. *Las Metodologías Ágiles y las Arquitecturas de Software.* Mexico : Coloquio Nacional de Investigación en Ingeniería de Software y Vinculación Academia-Industria, 2010.

VP Inc. 2013. Visual Paradigm. [Online] VP Inc., 2013. [Cited: febrero 7, 2013.] http://www.visual-paradigm.com/product/vpuml/.

Wake, W.C. 2002. *Extreme Programming Explored.* s.l. : Addison-Wesley, 2002.

Bibliografía Consultada

Aho, Alfred V., et al. 2006. *Compilers: Principles, Techniques, and Tools [Second Edition].* Stanford : Pearson Education Inc., 2006. 0-321-48681-1.

Alchin, Marty. 2009. *Pro Django.* New York : Apress, 2009. 978-1-4302-1048-1.

Ashworth, Stuart and Duncan, Andrew. 2012. *Ext JS 4 Web Application Development Cookbook.* United Kingdom : Packt Publishing Ltd, 2012. 978-1-84951-686-0

Auer, K and Miller, R. 2002. *Extreme Programming Applied.* s.l. : Addison-Wesley, 2002.

Battaner Arias, Paz. *Diccionario del uso del Español de América y España.* Barcelona : SpesEditorial, 2008. 84-8332-483-0

Bradenbaugh, Jerry. 2000. *Aplicaciones JavaScript.* Madrid : Anaya Multimedia, 2000. 84-415--1070-9.

Eguíliz Pérez, Javier. 2008. *Introducción a Ajax* s.l. :, 2008

Freeman , Elisabeth and Freeman, Eric. 2004. *Head First Design Patterns.* s.l. : O'Relly, 2004.

Gonzales Duque, Raúl. 2010. *Python para todos.* España : s.n., 2010.

Hayward, Jonathan. 2011. *Django JavaScript Integration: AJAX and jQuery*. United Kingdom : Packt Publishing, 2011. 978-1-849510-34-9.

Holovaty, Adrian and Kaplan-Moss, Jacob. 2009. *The Definitive Guide to Django [Second Edition].* New York : Apress, 2009. 978-1-4302-1937-8.

MacCaw, Alex. 2012. *The Little Book on CoffeeScript*. USA : O'Reilly, 2012. 978-1-449-32105-5.

Morrison, Michael. 2008 *Head First JavaScript* USA : O'Reilly, 2008. 0-596-52774-8

Powers, Shelley. 2010 *JavaScript Cookbook*. USA : O'Reilly, 2010. 978-0-596-80613-2.

Ravi, Sauman. 1986. *Compilers: principles, techniques, and tools.* Boston, USA : Addison Wesley .Inc, 1986.

Real Academia Española. 2012. Diccionario de la Lengua Española [Vigésima segunda edición]. [Online] Real Academia Española, 2012. [Cited: 6 16, 2013.] http://lema.rae.es/drae

Roesch, Martin and Green, Chris. 2012. *Snort User Manual 2.9.3.* s.l. : Sourcefire inc., 2012.

Scott, Charlie , Wolfe, Paul and Hayes, Bert. 2004. *Snort for Dummies.* Indianapolis : Wiley Publishing, Inc, 2004. 0-7645-6835-3.

Jeffries , Ron, Anderson , Ann and Hendrickson , Chet. 2001. *Extreme Programming Installed.* New jersey : Addison-Wesley, 2001. 0-201-70842-6.

Glosario de Términos

Antivirus: son programas cuyo objetivo es detectar y/o eliminar virus informáticos.

Aplicación web: es una página web especial, que tiene una base de datos asociada y que permite una mayor interacción del usuario. Una aplicación web puede ofrecer funcionalidades tales como: correo electrónico, gestión de contenido, comunicación con el cliente, entre otras.

Bits: es la unidad mínima de información empleada en informática, en cualquier dispositivo digital, o en la teoría de la información. Con él, podemos representar dos valores cuales quiera, como verdadero o falso, abierto o cerrado, blanco o negro, norte o sur, masculino o femenino, rojo o azul, etc. Basta con asignar uno de esos valores al estado de "apagado" (0), y el otro al estado de "encendido".

Bus (*hardware*): es un sistema digital que transfiere datos entre los componentes de una computadora o entre computadoras. Está formado por cables o pistas en un circuito impreso, dispositivos como resistores y condensadores además de circuitos integrados.

Cache: es un área especial de memoria que poseen los ordenadores. Funciona de una manera similar a como lo hace la memoria principal (RAM), pero es de menor tamaño y de acceso más rápido.

Comando: es una instrucción u orden que el usuario proporciona a un sistema informático, desde la línea de comandos (como una *shell*) o desde una llamada de programación.

Cortafuego (*firewall*): es una parte de un sistema o una red que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas. Se trata de un dispositivo o conjunto de dispositivos configurados para permitir, limitar, cifrar, descifrar, el tráfico entre los diferentes ámbitos sobre la base de un conjunto de normas y otros criterios.

Estructura de datos: es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación.

Ethernet: es un estándar de redes de área local, define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del modelo OSI.

FTP: es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (del inglés *Transmission Control Protocol*). Este permite que un equipo cliente se pueda conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

Modelo OSI: marco de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones. Es un lineamiento funcional para tareas de comunicaciones y, por consiguiente, no especifica un estándar de comunicación para dichas tareas. Sin embargo, muchos estándares y protocolos cumplen con los lineamientos este modelo.

Programación funcional: es un paradigma de programación declarativa basado en la utilización de funciones aritméticas que no maneja datos mutables o de estado. Enfatiza la aplicación de funciones, en contraste con el estilo de programación imperativa, que enfatiza los cambios de estado. La programación funcional tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los 1930s para investigar la definición de función, la aplicación de las funciones y la recursión.

Programación Imperativa: en contraposición a la programación declarativa es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

Protocolos de red: conjunto de normas *standard* que especifican el método para enviar y recibir datos entre varios ordenadores. Es una convención que controla o permite la conexión, comunicación, y transferencia de datos entre dos puntos finales.

Protocolo HTTP: el Protocolo de Transferencia de HiperTexto (Hypertext Transfer Protocol) es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes Web y los servidores HTTP.

Protocolo TCP/IP: es un modelo de descripción de protocolos de red, describe un conjunto de guías generales de diseño e implementación de protocolos de red específicos para permitir que un equipo pueda comunicarse en una red. TCP/IP provee conectividad de extremo a extremo especificando cómo los datos deberían ser formateados, direccionados, transmitidos, encaminados y recibidos por el destinatario. Existen protocolos para los diferentes tipos de servicios de comunicación entre equipos.

Proxy: es un programa o dispositivo que se encarga de interceptar las conexiones de red que un cliente hace a un servidor de destino, por varios posibles motivos como seguridad, rendimiento, anonimato, u otras.

Router: es un dispositivo que proporciona conectividad, su función principal consiste en enviar o encaminar paquetes de datos de una red a otra, es decir, interconectar subredes.

Servidor web: un servidor web es un programa informático que procesa una aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente.

Sitio web: es una colección de páginas de internet relacionadas y comunes a un dominio de Internet.

Sniffer: es un programa o dispositivo que captura todos los datos que pasan a través de una tarjeta de red.

TokenRing: es una arquitectura de red desarrollada con topología física en anillo y técnica de acceso de paso de testigo, usando un token que viaja alrededor del anillo.

Unix: es un sistema operativo portable, multitarea y multiusuario. Existen varias familias del sistema operativo UNIX, que han evolucionado de manera independiente a lo largo de los años. Cada familia se distingue no tanto por sus diferencias técnicas como por sus diferencias en propiedad intelectual. Se observa que todas las familias se han visto contaminadas, directa o indirectamente, por otras familias.

URL: es la cadena de caracteres con la cual se asigna una dirección única a cada uno de los recursos de información disponibles en la web, para ello combina el nombre del ordenador que proporciona la información, el directorio donde se encuentra, el nombre del archivo, y el protocolo a usar para recuperar los datos para que no se pierda alguna información sobre dicho factor que se emplea para el trabajo.

Virus informático: es un tipo de software que tiene por objetivo alterar el normal funcionamiento de la computadora, sin el permiso o el conocimiento del usuario.

Anexos

HISTORIAS DE USUARIO

Historia de Usuario							No. 03
Nombre	Modificar regla						
Usuario	Administrador						
Prioridad	media	**Complejidad**	alta	**Estimación** (**sem**)	1	**Tipo**	funcional
Dependencia	HU1, HU8						
Descripción							
Para el administrador sería factible poder modificar una regla que ya esté creada. Así podría cambiar cualquier campo, opción o dato de la misma sin tener que crear una desde el principio; o cambiar algo a causa de un ajuste que necesita hacer.							
Información Adicional							
El administrador modifica la regla con la ayuda del editor, por lo cual esta historia tiene cierta dependencia respecto a la HU8. Igualmente, el árbol debe estar formado previamente.							

Tabla 19 Descripción de la HU número 3

Historia de Usuario							No. 04
Nombre	Eliminar regla						
Usuario	Administrador						
Prioridad	media	**Complejidad**	baja	**Estimación** (**sem**)	0.5	**Tipo**	funcional
Dependencia	HU1						
Descripción							
Un administrador del sistema deberá contar con la posibilidad de suprimir una regla de un fichero, a través de la interfaz del sistema.							
Información Adicional							
El administrador modifica la regla con la ayuda de un menú contextual en el árbol, por lo cual esta historia tiene cierta dependencia respecto a la HU1.							

Tabla 20 Descripción de la HU número 4

Historia de Usuario							No. 05
Nombre	Leer regla						
Usuario	Administrador						
Prioridad	alta	**Complejidad**	alta	**Estimación** (**sem**)	2	**Tipo**	funcional
Dependencia							
Descripción							
El administrador debe contar con algún componente intermedio que le permita al sistema mapear las reglas desde los ficheros, para luego poder mostrar sus datos de manera más estructurada. Esto debe ocurrir de forma dinámica, debido a que los cambios en el sistema deben reflejarse en el fichero que se está procesando.							
Información Adicional							

Tabla 21 Descripción de la HU número 5

Historia de Usuario							No. 06
Nombre	Listar reglas						
Usuario	Administrador						
Prioridad	media	**Complejidad**	baja	**Estimación** (**sem**)	0.5	**Tipo**	funcional
Dependencia	HU1						
Descripción							
A partir de que las reglas se guardan en archivos, el administrador del sistema desea contar con algún modo de poder visualizar las reglas correspondientes a un fichero.							
Información Adicional							
La función de esta HU podría aprovecharse en conjunto con el árbol, el cual podría listar las reglas que pertenecen a un fichero determinado.							

Tabla 22 Descripción de la HU número 6

Historia de Usuario							No. 07
Nombre	Cargar fichero de reglas						
Usuario	Administrador						
Prioridad	alta	**Complejidad**	media	**Estimación** (**sem**)	1	**Tipo**	funcional
Dependencia							
Descripción							
Se desea que el sistema sea capaz de cargar las reglas que se encuentran en un fichero.							
Información Adicional							

Tabla 23 Descripción de la HU número 7

Historia de Usuario							No. 09
Nombre	Orientación a íconos						
Usuario	Administrador						
Prioridad	media	**Complejidad**	baja	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
Las funcionalidades principales del sistema estarán orientadas a iconos para un mayor reconocimiento por parte del usuario.							
Información Adicional							

Tabla 24 Descripción de la HU número 9

Historia de Usuario	No. 10
Nombre	Tiempo de reparación menor de 7 días

Usuario	Administrador						
Prioridad	baja	**Complejidad**	-	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
El tiempo de reparación del sistema en caso de fallo tiene un máximo de 7 días.							
Información Adicional							

Tabla 25 Descripción de la HU número 10

Historia de Usuario						**No.**	11
Nombre	Disponibilidad de información y funcionalidades						
Usuario	Administrador						
Prioridad	baja	**Complejidad**	baja	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
Los servicios del sistema estarán disponibles durante las 24 horas de los 7 días de la semana, ejecutándose en servidores dedicados.							
Información Adicional							

Tabla 26 Descripción de la HU número 11

Historia de Usuario						**No.**	13
Nombre	Diseño de pocas entradas						
Usuario	Administrador						
Prioridad	baja	**Complejidad**	baja	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
Diseño sencillo, con pocas entradas, donde no sea necesario mucho entrenamiento para utilizar el sistema.							
Información Adicional							

Tabla 27 Descripción de la HU número 13

Historia de Usuario						**No.**	14
Nombre	Basado en arquitectura cliente-servidor						
Usuario	Administrador						
Prioridad	baja	**Complejidad**	media	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
El producto de software final debe diseñarse sobre una arquitectura cliente-servidor.							
Información Adicional							

Tabla 28 Descripción de la HU número 14

Historia de Usuario							No. 17
Nombre	Normativas legales de la Universidad de las Ciencias Informáticas						
Usuario	Administrador						
Prioridad	baja	**Complejidad**	baja	**Estimación** (**sem**)	<1	**Tipo**	propiedad
Dependencia							
Descripción							
Como producto, el sistema de detección de intruso se distribuye amparado bajo las normativas legales establecidas en el registro comercial emitido por las entidades jurídicas de la Universidad de las Ciencias Informáticas.							
Información Adicional							

Tabla 29 Descripción de la HU número 17

TARJETAS CRC

AbstracFile	
Clase abstracta para modelar un archivo y definir el comportamiento de otras más especializadas que son: File y Directory.	
Superclase	
Subclase	Directory, File
Atributos	
Nombre	Descripción
id	Identificador de la regla
name	Nombre de la regla
url	Dirección de la regla
Responsabilidad	**Colaborador**
Conocer si un fichero contiene reglas.	

Tabla 30 CRC clase Rule

Directory	
Clase especializada que modela un directorio del sistema de archivos.	
Superclase	AbstractFile
Subclase	
Atributos	
Nombre	Descripción

files	Lista de ficheros contenidos en el directorio modelado
Responsabilidad	**Colaborador**
Agregar Ficheros al directorio	
Devolver ficheros	

File	
Clase especializada que modela un fichero de regla, generalmente contenido dentro de un directorio.	
Superclase	AbstractFile
Subclase	
Atributos	
Nombre	Descripción
rules	Lista de reglas contenidas dentro del fichero
fileinfo	Se le asigna todo el texto dentro del fichero que no corresponde a una regla de detección
Responsabilidad	**Colaborador**
Cargar reglas del fichero	Rule
Encontrar regla	
Agregar regla al fichero	Rule
Habilitar regla	Rule
Borrar regla	
Deshabilitar Regla	Rule
Salvar archivo de regla	

Tabla 31 CRC clase Directory

Tabla 32 CRC clase File

Tabla 33 CRC clase JSONcodec

JSONcodec	
Clase auxiliar para codificar y decodificar objetos JSON.	
Superclase	
Subclase	
Atributos	
Nombre	Descripción
Responsabilidad	**Colaborador**
Devolver ítems en json	

Devolver reglas en json	
Convertir regla a json	
Devolver árbol de directorios en json	
Devolver árbol de reglas en json	

IDsClass	
Clase auxiliar para la generación de los identificadores de directorios, ficheros y reglas.	
Superclase	
Subclase	
Atributos	
Nombre	Descripción
directorioId	Identificador de directorio
fileId	Identificador de fichero de regla
ruleId	Identificador de regla
Responsabilidad	**Colaborador**
Devolver identificador de directorio	
Devolver identificador de fichero	
Devolver identificador de regla	
Reiniciar los identificadores	

Tabla 34 CRC clase IDsClass

Exceptions	
Contiene clases dentro para modelar excepciones que se pueden suscitar en tiempo de ejecución	
Superclase	
Subclase	
Atributos	
Nombre	Descripción
Responsabilidad	**Colaborador**
Lanzar excepción de posición inválida	
Lanzar excepción de camino incorrecto	
Lanzar excepción de camino de regla incorrecto	

Tabla 35 CRC clase Exceptions

CONJUNTO DE NO TERMINALES DE LA GRAMÁTICA

<regla>
<simple>
<mas>
<cabecera>
<accion>
<protocolo>
<origen>
<direccion>
<puerto>
<sentido>
<otroPuerto>
<listaOpciones>
<unaOpcion>
<masOpciones>
<opcion>
<msg>
<ref>
<idURL>
<gid>
<sid>
<rev>
<priority>
<meta>
<cuerpoMeta>
<unCuerpoMeta>
<masCuerpoMeta>
<classname>
<opciones>
<destino>
<class>
<listaPuerto>

CONJUNTO DE TERMINALES DE LA GRAMÁTICA

Acciones y Palabras reservadas

<tk_alert, “alert”,#>
<tk_log, “log”, #>
<tk_pass, “pass”, #>
<tk_activate, “activate”, #>
<tk_dynamic, “dynamic”, #>
<tk_drop, “drop”, #>
<tk_reject, “reject, #”>
<tk_sdrop, “sdrop”, #>
<tk_any, “any”, #>

Protocolo

<tk_icmp, “icmp”, #>
<tk_tcp, “tcp”, #>
<tk_udp, “udp”, #>
<tk_ip, “ip”, #>

Literales

<tk_ipd, “#.#.#.#”, # >
<tk_cidr, “#.#.#.#/#”, #>
<tk_entero, “#”, #>
<tk_cadena, “[a|…|z|A|…|Z|0|…|9]”, #>
<tk_id, “([a|…|z|A|…|Z])⁺(a|…|z|A|…|Z|0|…|9)”, #>

Símbolo

<tk_forward, “->”, #>
<tk_bothsides, “<>” , #>
<tk_parent-open, “(”, #>
<tk_parent-close, “)”, #>
<tk_cor-open, “[”, #>
<tk_cor-close, “]”, #>
<tk_ptcoma, “;”, #>
<tk_dospt, “:”, #>
<tk_coma, “,”, #>
<tk_or, “|”, #>

Variables globales

<tk_http_ports, “$HTTP_PORTS”, #>
<tk_http_services, “$HTTP_SERVERS”, #>
<tk_external_net, “$HTTP_EXTERNAL_NET”, #>

Opciones generals

<tk_msg, “msg”, #>
<tk_reference, “reference”,#>
<tk_gid, “gid”, #>
<tk_sid, “sid”, #>
<tk_rev, “rev”, #>
<tk_classtype, “classtype”, #>
<tk_priority, “priority”, #>
<metadata, “metadata”, #>

Valores de opciones definidas

<tk_bugtraq, “bugtraq”, #>
<tk_cve, “cve”, #>
<tk_nesus, “nesus”, #>
<tk_arachnids, “arachnids”, #>
<tk_mcafee, “mcafee”, #>
<tk_osvdb, “osvdb”, #>
<tk_url, “url”, #>
<tk_ tk_engine_shard, “engine_shard” , #>
<tk_soid, “soid” , #>
<tk_service_http, “service /s http”, #>
<tk_attempted_admin, “attempted_admin”, #>
<tk_attempted_user, “attempted_user”, #>
<tk_inappropiate_content, “inappropriate_content”, #>
<tk_policy_vilolation, “policy_vilolation”, #>
<tk_shellcode_detect, “shellcode_detect”, #>
<tk_successful_admin, “succeful_admin”, #>
<tk_successful_user, “succeful_user”, #>
<tk_trojan_activity, “trojan_activity”, #>
<tk_unsuccessful_user, “unsuccessful_user” , #>
<tk_web_application_attack, “web_application_attack”, #>
<tk_attempted_dos, “attempted_dos”,#>
<tk_attempted_recon, “attempted_recon”,#>
<tk_bad_unknown, “bad_unknown”, #>
<tk_default_login_attempt, “default_login_attempt”, #>
<tk_denial_of_service, “denial_of_service”, #>
<tk_misc_attack, “misc_attack”, #>
<tk_non_standard_protocol, “non_standard_protocol”, #>
<tk_rpc_portmap_decode, “rpc_portmap_decode”, #>
<tk_successful_dos, “successful_dos”, #>
<tk_successful_recon_largescale, “successful_recon_largescale”, #>
<tk_successful_recon_limited, “successful_recon_limited”, #>
<tk_suspicious_filename_detect, “suspicious_filename_detect”, #>
<tk_suspicious_login, “suspicious_login”, #>
<tk_system_call_detect, “system_call_detect”, #>

```
<tk_unusual_client_port_connection, “unusual_client_port_connection”,#>
<tk_web_application_activity, “web_application_activity”, #>
<tk_icmp_event, “icmp_event”, #>
<tk_misc_activity, “misc_activity”, #>
<tk_network_scan, “network_scan”, #>
<tk_not_suspicious, “not_suspicious”, #>
<tk_protocol_command_decode, “protocol_command_decode”, #>
<tk_string_detect, “string_detect”,#>
<tk_unknown, “unknown”, #>
<tk_tcp_connection, “tcp_connection”, #>
```

CASOS DE PRUEBA

Modificar regla		**HU3**
Iteración	3	
Crítico	Si	
Función	El editor debe permitir modificar una regla de detección.	
Precondiciones	Debe haber una regla activa en el editor, esto se consigue haciendo doble clic en una regla del árbol de ficheros.	
Acción	**Datos**	**Resultado Esperado**
Escribir	Datos estrados por el usuario	El editor muestra los datos.
Clic en el botón de guardar	Datos estrados por el usuario	Notificación de regla guardada

Tabla 36 CP historia de usuario 3

Leer regla		**HU5**
Iteración	3	
Crítico	Si	
Función	Se debe leer una regla del archivo.	
Precondiciones	El árbol de ficheros debe estar cargado.	
Acción	**Datos**	**Resultado Esperado**
Doble clic en una regla del árbol		Se muestra todo el contenido de la regla, en el editor
Doble clic en una regla recién añadida		Se muestra en el editor la indicación de añadir contenido a la nueva regla

Tabla 37 CP historia de usuario 3

Listar reglas		**HU6**
Iteración	3	
Crítico	No	
Función	Debe permitir mostrar una lista de las reglas que pertenecen a un fichero.	
Precondiciones	El árbol de ficheros debe estar cargado.	

Acción	Datos	Resultado Esperado
Doble clic al fichero de regla en el árbol	Id de fichero	Se muestran indexadas las reglas que pertenecen al fichero

Tabla 38 CP historia de usuario 6

Cargar fichero de reglas		HU7
Iteración	3	
Crítico	Si	
Función	El fichero de reglas debe ser mapeado completamente en una estructura de datos del sistema.	
Precondiciones	Haber entrado la dirección URL del origen del archivo de configuración de Snort.	
Acción	**Datos**	**Resultado Esperado**
Clic en “Aceptar” en la ventana de entrada de la dirección URL	URL	Se muestra le árbol de ficheros y reglas.
Clic en un fichero de regla	ID del fichero	Se despliega una lista de regalas pertenecientes al fichero, indexadas bajo este.

Tabla 39 CP historia de usuario 7

Editor de reglas con codificación inteligente		HU8
Iteración	3	
Crítico	Si	
Función	El edito debe brindar asistencia al usuario en la modificación y creación de las reglas.	
Precondiciones	Existir una regla activa en la barra de etiquetas.	
Acción	**Datos**	**Resultado Esperado**
Escribir en el editor un regla incorrecta, con elementos que están bien		Resaltado de la palabra incorrecto. Las palabras claves se muestran en azul
Oprimir la tecla control y espacio		El sistema muestra en una ventana las palabras esperadas, permitiendo el completamiento de la regla.

Tabla 40 CP historia de usuario 8