3 Modelado y desarrollo del IDS

En el presente capítulo, se enumeran los elementos de software a utilizar en el diseño, modelado y desarrollo del IDS, posteriormente, se hace una descripción paso a paso del desarrollo del IDS, se muestran algunos diagramas de flujo del IDS a desarrollar y sus respectivos algoritmos, y finalmente se exhibe el diseño de la ventana preliminar del sistema.

3.1 Elementos a utilizar en el diseño, modelado y desarrollo

A continuación se describen cada uno de los elementos necesarios para llevar a cabo el proyecto, así como las cualidades por los que fueron elegidos.

3.1.1 A cerca de Microsoft Visual C# 2008

Microsoft Visual C# es un lenguaje simple pero poderoso dirigido principalmente a desarrolladores que crean aplicaciones utilizando Microsoft .NET Framework. Recibe muchas de las mejores características de C++ y Microsoft Visual Basic pero pocas de las contradicciones y anacronismos, resultando un lenguaje más lógico y más limpio. Con la llegada de C# 2.0 en 2005, se agregaron varias características nuevas importantes, incluyendo procedimientos genéricos, iteradores y anónimos. C# 3.0, que se encuentra disponible como parte de Microsoft Visual Studio 2008, agrega más características adicionales tales como, por ejemplo, métodos de extensión, cálculos lambda, y, la más famosa de todas, la habilidad *Language Integrated Query*, o LINQ. El entorno de desarrollo proporcionado por Visual Studio 2008 hace estas características fáciles de utilizar, y la multitud de asistentes mejoras nuevas incluidas en Visual Studio 2008 pueden perfeccionar en gran medida su productividad como un desarrollador [52].

3.1.2 SharpPcap

SharpPcap es una librería de captura de paquetes para el ambiente .NET, basado en el famoso componente WinPcap. El propósito de esta librería es proporcionar un API para captura, inyección, análisis y construcción de paquetes utilizando cualquier lenguaje .NET tal como C# y Visual Basic.NET [54]. La siguiente lista, enumera las características que actualmente soporta SharpPcap:

- Enumera y muestra detalles acerca de la interfaz física de red en una computadora con Windows [54].
- Captura paquetes de bajo nivel a través de la interfaz seleccionada [54].
- Analiza los paquetes de los siguientes protocolos: Ethernet, ARP, IP, TCP, UDP, ICMP e IGMP [54].
- Inyecta paquetes de bajo nivel a la red en la interfaz seleccionada [54].
- Maneja (lee y escribe) archivos de paquetes capturados fuera de línea [54].
- Inyecta paquetes utilizando colas de envío (Send Queues) [54].
- Recolecta estadísticas de la red de una interfaz dada [54].

3.1.3 Reglas de Snort

Una de las funciones más valoradas de Snort es la capacidad de que los usuarios puedan escribir sus propias reglas, además de una larga base de datos de reglas que Snort contiene por default, administradores de IDS pueden tomar ventajas de dicha capacidad para desarrollar sus propias reglas en vez de tener que depender de una agencia externa, vendedor o administrador para actualizaciones en caso de que un nuevo ataque o exploit sea descubierto, los administradores de Snort pueden escribir sus propias reglas para todo tráfico que ellos consideren anómalo y comparar notas con una gran comunidad de escritores de reglas de Snort en internet. Esto permite capacidades imprecdentes en cuanto a velocidad de actualización y personalización. Una regla es un conjunto de instrucciones diseñadas para tomar un paquete del tráfico de la red y comparar un patrón específico, para entonces tomar una acción cuando la regla coincida con el paquete de tráfico [50].

Base de datos de las reglas de ataques

El NIDS implementado en este proyecto será capaz de analizar el tráfico de la red mediante la base de datos de ataques de Snort. Un buen NIDS necesita una buena y extendida base de datos de firmas. Pero para crear y mantener una base de datos actualizada que sea suficientemente buena para este proyecto, implica un gran esfuerzo. Por esa razón este NIDS, será capaz de procesar las reglas basadas en Snort, el cual cuanta con una gran cantidad de reglas desarrolladas y actualizadas, además cuenta con amplia documentación y facilita la creación de nuevas reglas [12].

El lenguaje usado por Snort es flexible y potente, basado en una serie de normas que servirán de guía para la escritura de las reglas [40]. Dentro de estas normas tenemos:

- La descripción de cada regla
- Cabecera
- Opciones
- Uso de preprocesadores

Las reglas de Snort se pueden dividir en dos secciones lógicas: cabecera y opciones como se puede observar en la Tabla 3.1:

- La cabecera contiene la acción de la regla en sí, protocolo, IPs, máscaras de red, puertos origen y destino y destino del paquete o dirección de la operación.

- La sección opciones contiene los mensajes y la información necesaria para la decisión a tomar por parte de la alerta en forma de opciones.

Tabla 3.1.- Secciones lógicas en que se divide una regla de Snort.

Cabecera	**Opciones**
Acción	Mensaje
Protocolos involucrados	Opciones de decisión
Direcciones IP	
Números de puerto	
Dirección de la operación	

|| CABECERA ~- Acción ~- Protocolos involucrados ~- Direcciones IP ~- Números de puerto ~- Dirección de la operación || OPCIONES ~- Mensaje ~- Opciones de decisión ||

Ejemplo de regla Snort para alertar de un escaneo Nmap del tipo TCP ping:

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Escaneo ping con nmap";flags:A;ack:0; reference:arachnids,28;classtype:attempted-recon; sid:628; rev:1;)

Los elementos del ejemplo anterior son:

Cabecera:

- **Acción de la regla**: alert
- **Protocolo**: TCP
- **Dirección IP origen**: $EXTERNAL_NET (toda la red)
- **Puerto IP origen**: any (cualquiera)
- **Dirección IP destino**: $HOME_NET (toda nuestra red)
- **Puerto IP destino**: any (cualquiera)
- **Dirección de la operación**: -> (puede ser ->, <-, <>)

Opciones

- **Mensaje**: msg
- **Opciones**: flags:A;ack:0; reference:arachnids…(1)

Desglosando las opciones:

- **flags:A** Establece el contenido de los flags ó banderas TCP, en este caso ACK (puede tener varios valores y operadores).
- **ack:0** Caso particular para valor ACK=0, es el valor que pone Nmap para TCP ping scan.
- **reference:arachnids,28** Referencia un a un Advisory, alerta tipo Bugtrac, etc.

- **classtype:attempted-recon** Categoría de la alerta según unos niveles predefinidos y prioridades**.**
- **sid:628** Identificación única para esta regla Snort según unos tramos determinados.
- **rev:1** Identificación de la revisión o versión de la regla.

Las reglas Snort se ubican en archivos "***.rules***" (Snort rules). A continuación se muestra un ejemplo de uno de estos archivos:

```
# (C) Copyright 2001,2002, Martin Roesch, Brian Caswell, et al.
# All rights reserved.
# $Id: virus.rules,v 1.16 2002/08/18 20:28:43 cazz Exp $
#------------
# VIRUS RULES
#------------
#
# NOTE: These rules are NOT being actively maintained.
#
#
# If you would like to MAINTAIN these rules, e-mail
# snort-sigs@lists.sourceforge.net
#

alert tcp any 110 -> any any (msg:"Virus - SnowWhite Trojan Incoming"; content:"Suddlently"; sid:720; classtype:misc activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible pif Worm"; content: ".pif"; nocase; sid:721; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible NAVIDAD Worm"; content: "NAVIDAD.EXE"; nocase; sid:722; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "myromeo.exe"; nocase; sid:723; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "myjuliet.chm"; nocase; sid:724; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "ble bla"; nocase; sid:725; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "I Love You"; sid:726; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "Sorry... Hey you !"; sid:727; classtype:misc-activity; rev:3;)

alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "my picture from shake-beer"; sid:728; classtype:misc-activity; rev:3;)
```

Lo importante de este archivo de texto plano, son las reglas, el resto (con la marca #) se refiere a un título informativo. Dichos archivos comúnmente se almacenan en el directorio Rules de Snort [40].

3.2 Modelado del IDS UDLAP

El siguiente proyecto es un programa creado con el objetivo de detectar accesos no autorizados a una computadora o a una red, ya que estos accesos se pueden tratar de ataques de hackers. El funcionamiento del IDS-UDLAP se basa en el análisis pormenorizado del tráfico de la red, el cual al entrar al programa, es comparado con firmas de ataques conocidos en tiempo real. En resumen, el IDS-UDLAP carga las reglas de Snort a memoria, las cuales se organizan en una tabla creada mediante el algoritmo de "Aho-corasick", posteriormente, el programa detecta y enlista los dispositivos de captura de tráfico disponibles en el sistema en donde se encuentra alojado el programa, el usuario selecciona el dispositivo de su preferencia, mediante el dispositivo seleccionado comienza la captura de paquetes, el sistema divide los paquetes de acuerdo a su protocolo, si encuentra un paquete fragmentado lo reensambla y a continuación lo normaliza, es decir, por ejemplo, si el paquete se encuentra escrito en hexadecimal, lo traduce a ASCII. Finalmente se compara el paquete con la tabla creada por el algoritmo "Aho-Corasick", si el paquete coincide con alguna regla, éste se marca como alerta y se agrega al reporte. Paralelamente todos los paquetes son mostrados en la pantalla del IDS. Éstos son los pasos a seguir para el diseño y programación:

1.- Cargar en memoria las reglas de Snort.
2.- División de las reglas para crear grupos.
3.- Creación de una tabla mediante el algoritmo de Aho-Corasick.
4.- Obtención de la lista de dispositivos.
5.- Apertura de un adaptador.
6.- Captura de paquetes.
7.- División de paquetes de acuerdo a su protocolo.
8.- Reensamblado de paquetes.
9.- Normalización de paquetes.
10.- Creación de una tabla de datos que despliega los paquetes capturados.
11.- Comparación de paquetes con la tabla creada por el algoritmo Aho-Corasick.
12.- Envío de resultados a la tabla de datos.
13.- Envío de alertas a la ventana de resumen.

Más adelante se describirán a detalle cada uno de los elementos anteriores. En la Figura 3.1 se muestra a detalle el funcionamiento del IDS-UDLAP.

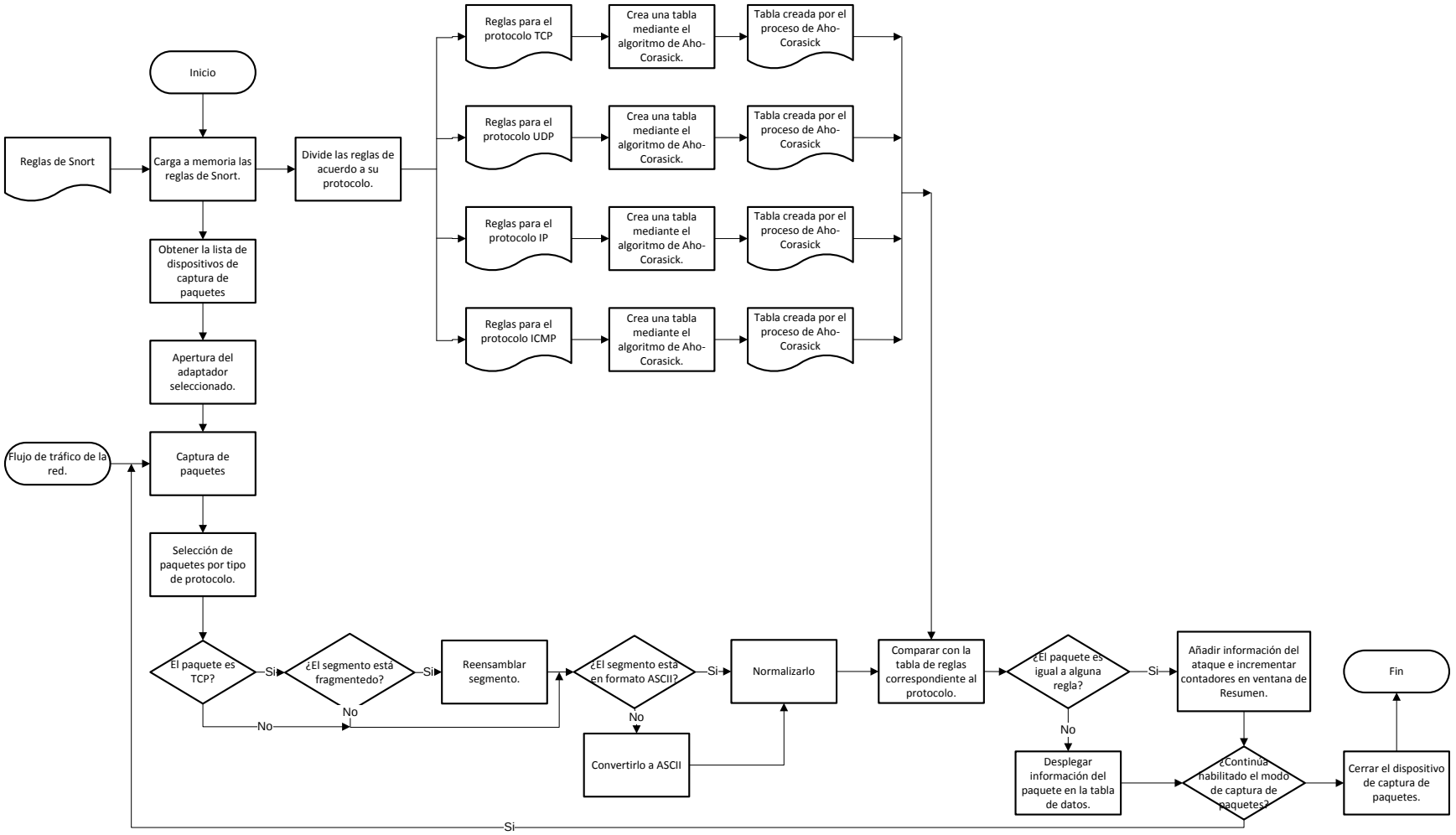

Figura 3.1.- Diagrama de flujo del IDS UDLAP

3.2.1 Cargar a memoria las reglas de Snort.

El programa tiene la opción de que el usuario pueda seleccionar el directorio en donde se encuentran las reglas, una vez que el usuario seleccionó el directorio, el programa busca los archivos con terminación ".rules", para cada archivo con dicha terminación lee cada línea, y cada vez que encuentra la palabra "*alert*" sube la línea a memoria. Una vez que ha terminado de leer cada archivo del directorio hace un conteo de cada línea con el fin de mostrar en la caja de mensajes de la ventana principal el número total de reglas a procesar.

3.2.2 División de las reglas para crear grupos.

Una vez que las reglas son cargadas en memoria, el sistema las clasifica de acuerdo a sus respectivos protocolos, es decir, crea 4 grupos: TCP, UDP, IP e ICMP. Después de haber sido creados dichos grupos, se organizan de acuerdo a ciertos campos: TCP y UDP se ordenan de acuerdo a su "dirección de origen" y "dirección de destino", IP se ordena de acuerdo a su campo "tipo" y por último las reglas ICMP se ordenan de acuerdo a su campo "protocolo".

3.2.3 Creación de una tabla mediante el algoritmo de Aho-Corasick.

El IDS-UDLAP utiliza el algoritmo de comparación de patrones llamado "Aho-Corasick", éste algoritmo resulta muy rápido para encontrar patrones, aun que la desventaja es que requiere bastante RAM [44].

El algoritmo de Aho-Corasick es un algoritmo de búsqueda de cadenas inventado por Alfred V. Aho y Margaret J. Corasick. El algoritmo crea una especie de diccionario de coincidencias que localiza elementos de un número finito de cadenas a partir de un texto de entrada [58]. Para adaptar este algoritmo al programa, la creación de éste se dividirá en tres partes:

- Creación de la máquina de estado.
- Implementación de enlaces mediante la función de error.
- Creación de una tabla con la información obtenida.

Creación de la máquina de estado.

Una vez creados los grupos de filas, se eliminan los espacios, el algoritmo Aho-Corasick, crea una máquina de estado basada en los grupos que resultan del paso anterior, la máquina de estado está compuesta por: un nodo raíz "estado disponible", nodos hijos y enlaces de la función de error para facilitar una búsqueda transversal. Para comenzar a construir la máquina de estado, se comienza con el estado predefinido que en este caso es el "estado disponible" el cual fungirá como nodo raíz, para agregar nuevas cadenas, se recorre la máquina de estado hasta encontrar el nodo que coincida con la nueva cadena, si éste no existe, cada carácter de la nueva cadena agrega un nodo a la máquina de estado a parir del "estado disponible", y por el contrario, si se encuentra un nodo que coincida con la nueva cadena, el resto de los caracteres de la cadena agregan nuevos nodos a partir de dicho nodo[44]. La Figura 3.2 muestra el proceso de construcción de la máquina de estado de Aho-Corasick.

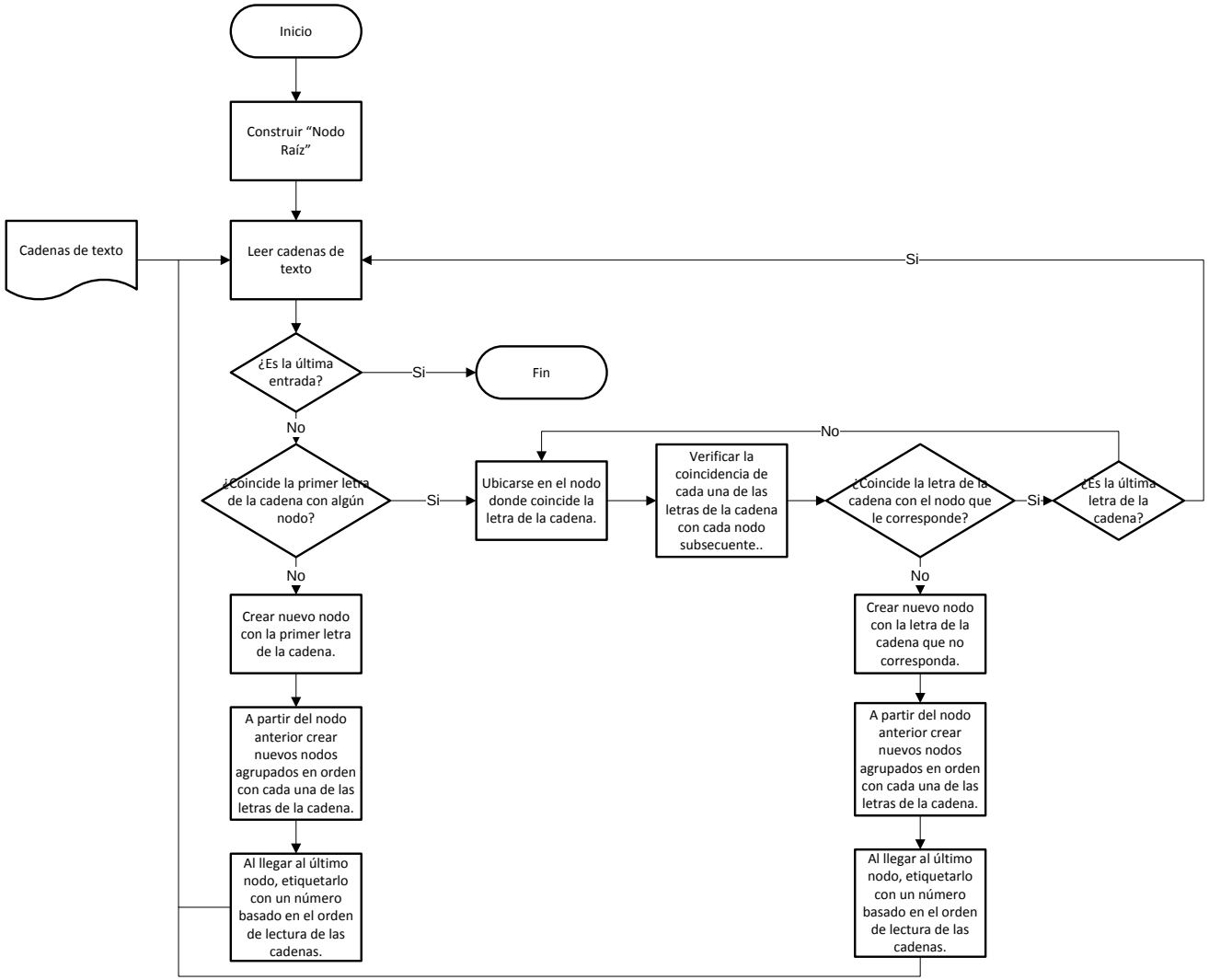

Figura 3.2.- Diagrama de creación de una máquina de estado mediante el algoritmo de Aho-Corasick

La Figura 3.3 muestra un pequeño ejemplo de máquina de estado con las cadenas {he, his, she, hers} (Nótese que es importante el orden en que las cadenas entran al sistema).

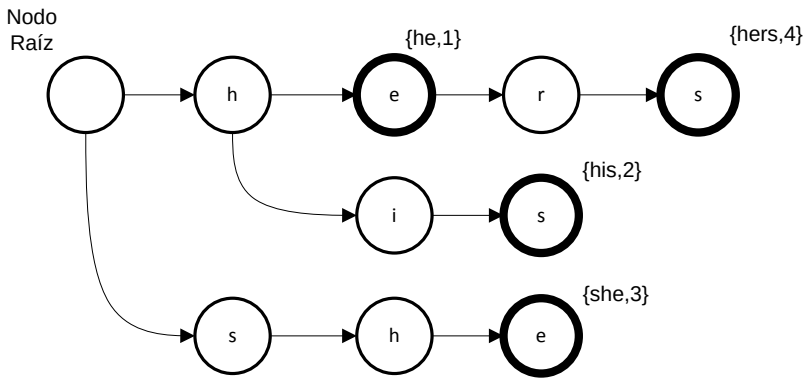

Figura 3.3.- Ejemplo de una máquina de estado con cadenas específicas.

El orden de las columnas también es importante ya que entre más cerca esté la columna del nodo raíz mayor es su grado de preferencia y se debe tomar en cuenta para la siguiente sección.

Implementación de enlaces mediante la función de error.

Cuando una cadena no encuentra coincidencia, es posible que el sufijo de la misma si encuentre. Para tratar estos casos, se hace lo siguiente: Una vez que la máquina de estado terminó de construir un árbol, se catalogan los nodos de manera que entre más cerca se encuentren del nodo raíz mayor es su grado. A continuación se leen cada uno de los nodos del árbol y si se encuentra un caracter que coincida con otro de mayor grado, se establece un enlace en dirección al de mayor grado [44]. La Tabla 3.2 muestra el proceso de agregar cadenas e implementación de enlaces mediante la función de error.

Tabla 3.2.- Pseudo-código para agregar cadenas.

```
Agregar cadena()
        estado = estado disponible
        for cada caracter en la nueva cadena encontrar coincidencia buscando en cada nodo
                if parte de la cadena existe en el árbol
                        agregar el resto de la cadena al árbol
                        asignar un número de id al último nodo de la cadena
                else
                        comenzar desde el estado disponible y agregar el resto de la cadena
                        asignar un número de id al último nodo de la cadena
        end for
        recorrer el árbol cada vez que un nodo coincida con otro de mayor grado, crear un
        enlace.
end
```

La Figura 3.4 muestra el proceso de implementación de enlaces mediante la función de error.

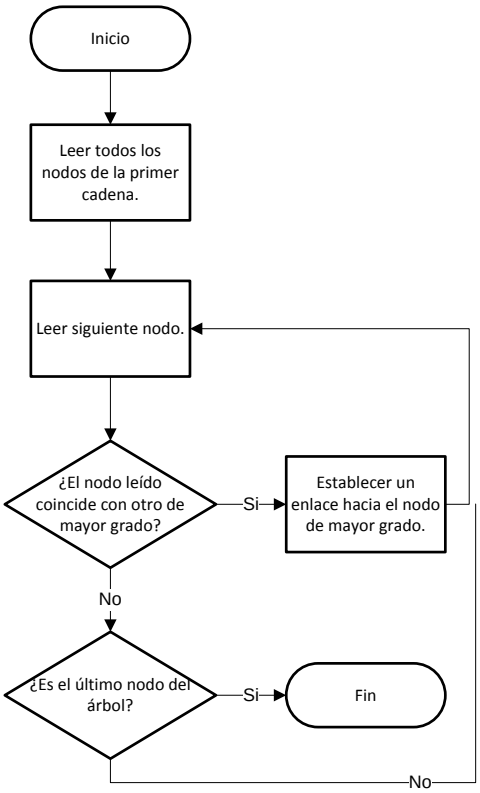

Figura 3.4.- Proceso de implementación de enlaces mediante la función de error.

Basándose en el ejemplo anterior ({he, his, she, hers}), en la Figura 3.5 y la Tabla 3.3 se muestra el resultado de la implementación de los enlaces mediante la función de error.

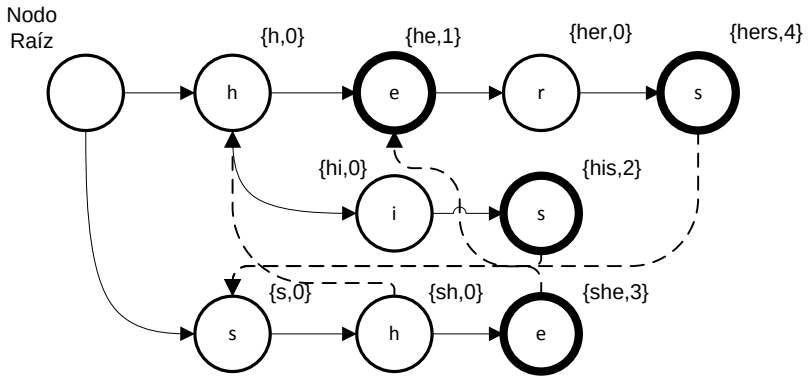

Figura 3.5.- Implementación de enlaces mediante la función de error.

Tabla 3.3.- Resultado de la implementación de enlaces mediante la función de error.

Nodo	{h,0}	{he,1}	{s,0}	{sh,0}	{she,3}	{hi,0}	{his,2}	{her,0}	{hers,4}
Apunta hacia	0	0	0	{h,0}	{he.1}	0	{s,0}	0	{s,0}

Creación de una tabla con la información obtenida.

Una vez que el programa termina de crear la máquina de estado, ésta es recorrida y se genera una tabla. Se utiliza la recursión para recorrer la estructura de datos y llenar la tabla de estado [44].

La tabla generada es una especie de traducción de la máquina de estado a un formato tabular. Las filas están indexadas por cada uno de los nombres de todos los nodos y las columnas por los caracteres de entrada. Cada elemento contiene un par de valores que se mostrarán en breve. La tabla tiene como título de columnas los caracteres de entrada de todas las cadenas de texto sin repeticiones, acomodados de manera que la primer columna corresponde a el segundo caracter de la primer cadena, la segunda columna corresponde al primer caracter de la misma cadena, la tercera al tercer caracter que entró al sistema, la cuarta al cuarto caracter que entró al sistema y así sucesivamente hasta tener cubiertos todos los caracteres de todas las cadenas, cuidando que no haya repeticiones, y como título para las filas, cada nodo del árbol, leyéndose por orden de introducción de las cadenas al sistema (comenzando por el nodo raíz) y yendo de mayor a menor grado (de izquierda a derecha) hasta llegar al final. Se retomará el ejemplo anterior con el objetivo de ilustrar el proceso de llenado. Éste se dividirá en 6 pasos:

El proceso de comparación se hace a partir de los elementos de la primer columna de la tabla. Se debe tomar en cuenta que todos los nodos siempre hacen referencia a aquellos nodos derivados directamente del nodo raíz, así como a aquellos que a su vez deriven directamente de ellos mismos.

- **Paso número 1:** Las celdas correspondientes a las columnas que hacen referencia a los nodos que derivan directamente del nodo raíz, se llenan con {[Nombre de la columna],0}. En este caso {h, 0} y {s, 0}.
- **Paso número 2:** El nodo que deriva directamente del que se está analizando tiene prioridad sobre el punto anterior. En este caso por ejemplo, el nodo “hers” comparado con el caracter “h” se debe llenar con {sh,0}, y en su caso también de manera similar los correspondientes a los nodos “her” y “hi” ambos comparados con el caracter “s”.
- **Paso número 3:** Cuando del elemento a comparar se deriva un enlace de error, se tomará en cuenta el nodo destino en vez del nodo origen. En este caso, el nodo “his” comparado con el nodo “h” se debe llenar con {sh,0} y en su caso de manera similar la casilla correspondiente a la comparación entre el nodo “s” con el caracter “h”.
- **Paso número 4:** Retomando los nodos que se derivan directamente del nodo que se está analizando, se llenan como por ejemplo los nodos “h” y “sh” comparados con el caracter “e”, se llenarían de la siguiente forma: {he, 1} y {she, 3}, etc.
- **Paso número 5:** Cuando el último caracter del nodo apunta mediante el enlace de error a otro nodo, tomará éste último el lugar del primero. En este caso cuando se compara “sh” con “i”, se llena con {hi,0} o “she” con “r”, se llena con {her,0}.
- **Paso número 6:** Todos los demás espacios se rellenan así: {-, 0}. Al final queda como la Tabla 3.4.

Tabla 3.4.- Tabla obtenida del proceso de Aho-Corasick.

	e	h	i	s	r
Nodo raíz	{-,0}	{h,0}	{-,0}	{s,0}	{-,0}
h	{he,1}	{h,0}	{-,0}	{s,0}	{-,0}
he	{-,0}	{h,0}	{-,0}	{s,0}	{her,0}
her	{-,0}	{h,0}	{-,0}	{hers,4}	{-,0}
hers	{-,0}	{sh,0}	{-,0}	{s,0}	{-,0}
hi	{-,0}	{h,0}	{-,0}	{his,2}	{-,0}
his	{-,0}	{sh,0}	{-,0}	{s,0}	{-,0}
s	{-,0}	{sh,0}	{-,0}	{s,0}	{-,0}
sh	{she,3}	{h,0}	{hi,0}	{s,0}	{-,0}
she	{-,0}	{h,0}	{-,0}	{s,0}	{her,0}

3.2.4 Obtención de la lista de dispositivos

Normalmente, lo primero que hace una aplicación basada en WinPcap es obtener una lista de adaptadores de red. SharpPcap proporciona la función GetAllDevices() para este propósito; esta función regresa una lista de dispositivos, en donde cada uno se muestra junto con información referente del mismo. En particular, los campos de PcapName y PcapDescription contienen el nombre del dispositivo junto con una descripción respectiva al dispositivo correspondiente. En el siguiente segmento de código, se muestra la manera en cómo se recupera una lista de adaptadores la cual posteriormente se muestra en pantalla, además de mostrar un error en caso de no encontrar adaptadores [54].

3.2.5 Apertura de un adaptador.

Después de haber obtenido la lista de dispositivos de red, SharpPcap tratará de utilizar el API de win32 IPHelper (siempre y cuando se encuentre presente) con el objetivo de recuperar mucho más información a cerca de el adaptador de red. Para este propósito se utilizará una clase de dispositivos de red “NetworkDevice”, la cual es una subclase de PcapDevice, que a su vez representa la tarjeta de red utilizada. Para cada dispositivo de red en la lista se puede combinar la informaciónde IPHelper con la de NetworkDevice.

La clase NetworkDevice contiene información fundamental del adaptador de red. Algo similar al comando “ipconfig” disponible en Windows NO, éste contiene información del IP (dirección IP, máscara de subred, puerta de enlace predeterminada), y la dirección MAC (dirección física) del adaptador, información de DHCP y WINS respectivamente.

3.2.6 Captura de paquetes.

La captura de paquetes es un elemento fundamental del sistema. Ésta se logra mediante una herramienta llamada WinPcap, en la siguiente sección se ampliará aún más esta información.

TCPDUMP

TCPdump es una herramienta de diagnóstico para redes TCP/IP basada en salida textual, que monitoriza los paquetes que entran y salen de una interfaz de red, y los presenta en formato legible, comúnmente denominado sniffer. No es intrínsecamente peligroso, y es de gran utilidad para los administradores de redes que necesiten ver el tráfico para poder buscar problemas en una red. Permite usar filtros mediante una expresión de búsqueda, y solo mostrará los paquetes cuya cabecera coincida con ella. También puede ser usada para la ingeniería inversa de protocolos de red. Para poder capturar todo el tráfico que llega a una interfaz de red, tanto como si está dirigida a ella (coinciden la MAC de destinatario y la de la tarjeta) como si no, es necesario colocarla en modo "promiscuo" (modo "monitor" para tarjetas Wi-Fi). Esto también permite que se pueda usar para capturar todo el tráfico que pasa por una red Ethernet basada en hubs, y cualquier otra red que no disponga de un dispositivo de enrutamiento inteligente como un switch o un router, con fines benignos o malignos. En este último caso, se suele combinar con ciertos tipos de ataque que hacen que sea útil también en redes que usen un switch [47].

Es importante mencionar que TCPdump no intenta recolectar el datagrama entero ilustrado en la Figura 3.6. La razón de esto es debido al volumen y a que el usuario normalmente está interesado en las porciones concernientes a la cabecera, las cuales por lo regular utilizan una longitud predeterminada, comúnmente de 68 bytes [47].

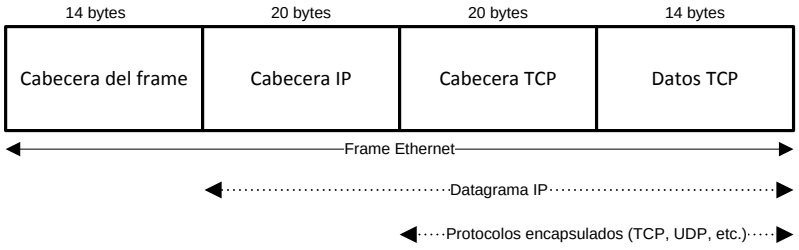

Figura 3.6.- Volumen de un datagrama entero.

Libpcap y WinPcap pueden ser utilizados por un programa para capturar los paquetes que viajan por toda la red y, en las versiones más recientes, para transmitir los paquetes en la capa de enlace de una red, así como para conseguir una lista de los interfaces de red que se pueden utilizar con el libpcap o WinPcap. Así mismo, libpcap y WinPcap son la captura del paquete y los motores de filtración de muchas herramientas de código abierto y comerciales de la red, incluyendo analizadores de protocolo, monitores de la red, sistemas de detección de intrusos en la red, programas de captura de las tramas de red (packet sniffers), generadores de tráfico y puesta a punto de la red. En este caso será utilizado para trabajar para Windump [18].

Como WinDump es sólo una versión de Windows de TCPdump, el uso de TCPdump y WinDump es prácticamente idéntico.

3.2.7 División de paquetes de acuerdo a su protocolo.

IDS-UDLAP divide los paquetes de entrada de acuerdo a su protocolo, esto es debido a que algunos de ellos vienen fragmentados, combinados o ambos. Por ejemplo TCP/IP. En este caso el sistema lee la cabecera con la que se adquiere el paquete y se clasifica dicho paquete de acuerdo a su protocolo. En el caso de ser IP, se verifica si está o no fragmentado y en caso de que no, se verifica si no es un paquete TCP/IP, en caso de ser así, se verifica que TCP tampoco se encuentre fragmentado. Con UDP se hace un procedimiento similar y para ICMP, los paquetes pasan directo al paso de normalización.

3.2.8 Reensamblado de paquetes.

Un paquete puede estar fragmentado. Antes de poder examinar una cadena dentro del paquete o determinar su tamaño exacto, es necesario defragmentarlo y ensamblar todos los fragmentos en un solo paquete de datos. Cada red impone algún tamaño máximo en sus paquetes. Esos límites tienen varias causas, algunas de ellas son las siguientes:

1.- Hardware (por ejemplo, el tamaño de un frame de Ethernet).

2.- Sistema operativo (por ejemplo, todos los buffers son de 512 bytes).

3.- Protocolos (Por ejemplo, el número de bits en el campo de la longitud del paquete).

El resultado de esos y otros factores es que los diseñadores de redes no son libres de escoger algún tamaño máximo de paquetes. El rango máximo para el cuerpo de un paquete es de 48 bytes en células ATM a 65,515 bytes en paquetes IP, aun que el tamaño del cuerpo de los paquetes en capas superiores suele ser mayor. Un problema obvio aparece cuando un paquete largo quiere viajar a través de una red cuyo tamaño máximo de paquetes es muy pequeño. Básicamente la única solución al problema es permitir que las compuertas rompan al paquete en fragmentos, enviando cada fragmento por separado. Para reensamblar paquetes, es necesario tener la información: si el paquete es parte de una secuencia de varios paquetes, de cuántos paquetes consta la secuencia, y por último es necesario saber cuándo se tienen todos los paquetes [55].

Debido a la manera en que viajan los datos en la red, los paquetes se dividen por protocolos. Los datagramas del protocolo IP pueden estar fragmentados, y al momento de juntarlos y obtener el datagrama original, se determina si éste a su vez lleva consigo paquetes fragmentados del protocolo TCP. Debido a esto, el proceso de reensamblado comienza con los paquetes IP. En este punto se ha llegado a una limitante de tiempo puesto que se comenzó a trabajar con el reensamblado de fragmentos TCP (un trabajo que es un tanto más laborioso y que requería más tiempo). Se le dio prioridad debido a que más del 90% de las reglas de Snort se basan en este protocolo. Aun que es preciso primero reensamblar los paquetes IP y posteriormente los TCP para poder concluir con mayor eficacia las conexiones abiertas.

Una entidad puede transmitir datos a otra entidad de tal forma que cada segmento es tratado independientemente de sus predecesores. Esto se conoce como transferencia de datos no orientada a conexión; un ejemplo es el uso del datagrama. Pese a que este modo es útil, una técnica igualmente importante es la transferencia de datos orientada a conexión, de la cual el circuito virtual, es un ejemplo [45].

3.2.9 Normalización de paquetes.

Uri.UnescapeDataString es un método de Microsoft Visual C# 2008 que convierte una cadena en su representación sin secuencias de escape. Muchos exploradores Web sustituyen los espacios de los identificadores URI por caracteres de escape "+"; sin embargo, el método UnescapeDataString no convierte los caracteres "+" en espacios porque este comportamiento no es estándar para todos los esquemas de URI [57].

3.2.10 Creación de una tabla de datos que despliega los paquetes capturados.

Para enviar a pantalla los paquetes procesados tanto los que contienen alertas como los demás, se utilizó un objeto llamado “DataTable”. DataTable es un objeto central de la biblioteca ADO.NET[1] . Entre los objetos que utilizan DataTable se incluyen DataSet y DataView. Si se va a crear un DataTable mediante programación, en primer lugar se debe definir su esquema agregando objetos DataColumn al DataColumnCollection (al que se obtiene acceso mediante la propiedad Columns). Para agregar filas con la información de los paquetes a DataTable, utiliza el método NewRow para devolver un nuevo objeto DataRow. El método NewRow devuelve una fila con el esquema de DataTable, tal como lo define el DataColumnCollection de la tabla. El número máximo de filas que puede almacenar un objeto DataTable es 16.777.216. DataTable también contiene una colección de objetos Constraint[2] que se pueden utilizar para asegurar la integridad de los datos. Hay muchos eventos DataTable que se pueden utilizar para determinar cuándo se realizan cambios en una tabla. Entre estos se incluyen los eventos RowChanged, RowChanging, RowDeleting y RowDeleted. Cuando se crea una instancia de DataTable, se establecen algunas propiedades de lectura y escritura en valores iniciales. Algunos de los campos de información de los paquetes agregados a DataTable son: dirección IP origen, dirección IP destino, puerto origen, puerto destino, etiqueta de tiempo, información del contenido del paquete, etc. [57].

3.2.11 Comparación de paquetes con la tabla creada por el algoritmo Aho-Corasick.

Para hacer coincidir el cuerpo de los paquetes, el sistema procesa el cuerpo del paquete caracter por caracter comenzando a partir del “estado disponible”. Para cada caracter en el paquete, se indexa el elemento {estado actual, caracter de entrada}. Si la entrada no existe en la tabla, el siguiente estado se convierte en disponible y el número identificador de la coincidencia se reinicializa a cero. Si una entrada existe en la tabla, la entrada de la tabla para el siguiente estado se con-

[1] ADO.NET es un conjunto de componentes de software que pueden ser usados por programadores con el objetivo de poder acceder a datos y a servicios de datos. Forma parte de la biblioteca de clases base de Microsoft.NET Framework.

[2] Son una restricción es una regla que se utiliza para mantener la integridad de los datos en el DataTable.

vierte en el estado actual y el número identificador de la coincidencia es verificado para ver si se encontró una cadena [44].

Por ejemplo, si se busca en el cuerpo del paquete "*go there*" en formato ASCII. Se comienza recorriendo la tabla restableciendo el estado actual a "disponible" y el número de identificación de la coincidencia en "cero". El paquete es procesado un caracter a la vez, comenzando con la letra "g". A la tabla se le indexa el elemento [disponible, g], al no existir columna para "g", el estado actual sigue siendo "disponible". Los siguiente dos caracteres son "o" y espacio" ", tampoco existen entradas en la tabla para ellos. El estado actual se mantiene en "disponible" y el número de identificación de la coincidencia sigue siendo "cero". El siguiente elemento es [disponible, t] para el cual la entrada de la tabla indica que el siguiente estado es "t" y el número de identificación de la coincidencia es "cero". El estado actual es cambiado a "t" y la tabla es indexada con [t, h]. La tabla es accesada y el estado actual se establece a "th". La tabla es indexada con [th, e] y el estado actual es cambiado a "the". El número identificador de la coincidencia es igual a tres, indicando que el paquete coincide con la cadena en la regla número tres. Algunos sistemas de detección se detienen cuando se encuentra la primera coincidencia y ejecutan una acción específica para dicha regla. IDS-UDLAP espera a que el paquete esté totalmente procesado para poder enviar una alerta. EL elemento [the, r] es el siguiente y el estado actual es cambiado a "ther". Finalmente, el último caracter del paquete es procesado y el elemento [ther, e] es utilizado para indexar la tabla. La entrada especifica un estado siguiente de "there" y un número identificador de coincidencia de cuatro. El paquete encuentra coincidencia en dos distintas cadenas que pertenecen a las reglas tres y cuatro [44]. La Tabla 3.5 muestra el proceso de comparación de paquetes con la tabla creada por el algoritmo de Aho-Corasick.

Tabla 3.5.- Pseudo-código para comparar paquetes contra la tabla creada con el algoritmo de Aho- Corasick.

```
comparar ()
        estado actual=disponible
        id de comparación = 0
        cadena = obtener (paquete de entrada) como ASCII
        for (i=1; i <= longitude de cadena (cadena); i++)
                if (encuentra un id de comparación) salir del ciclo
                if tabla[estado actual, cadena(i)]
                        {estado actual, id de comparación}= tabla[estado
                actual, cadena(i)]
                else
                        {estado actual, id de comparación}= {disponi-
                ble,0}
        end for
        return id de comparación
end
```

3.2.12 Envío de resultados a la tabla de datos.

Si las cadenas no coinciden, se repite el proceso hasta alcanzar el último caracter en el cuerpo del paquete, en este caso se envían los datos del paquete a la tabla de datos. Si una cadena coincide, se activa la alerta marcando el paquete y enviando la información del mismo a la tabla de datos junto con la alerta.

3.2.13 Envío de alertas a la ventana de resumen.

Una vez que las alertas son identificadas, se envían a la ventana de resumen en donde se clasifican por tipo de alerta y se lleva un contador en caso de que se repita alguna, de esta manera se despliega el contador y la información de la alerta.

3.3 Diseño preliminar de la ventana de salida

La Figura 3.7 muestra el diseño de la ventana principal del IDS a programar:

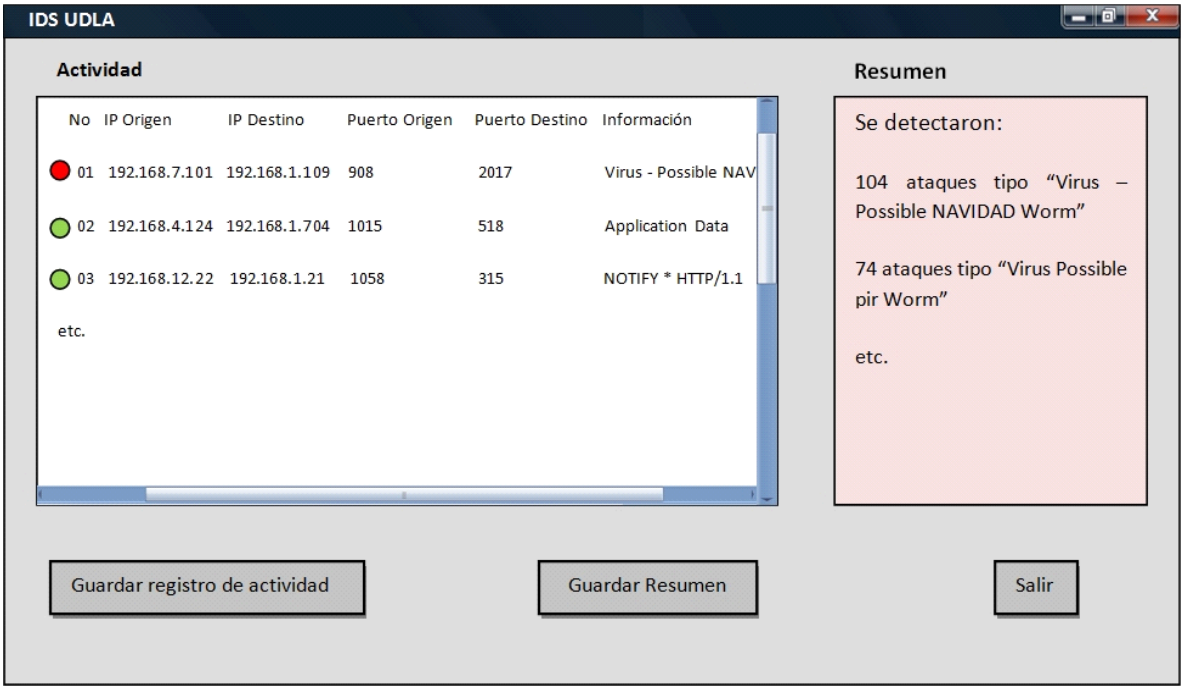

Figura 3.7.- Diseño preliminar de la ventana del IDS a programar.

En la ventana principal, se muestra el registro de la actividad lo más cercano posible a tiempo real, se tratará de crear un semáforo que indique si el paquete emite una alerta o no. En la ventana de resumen, se van almacenando el número de alertas así como información acerca de las mismas. Se puede guardar tanto el registro de la actividad como el resumen de ataques, para que posteriormente el usuario pueda acceder a los archivos de algún momento determinado o incluso imprimirlos.

3.4 Discusión

Se puede desprender una discusión acerca de qué técnica es la más adecuada para aplicar en la implementación de un IDS que detecte un ataque a partir de patrones previos o incluso, a partir de un proceso de inferencia propio. Algunas técnicas que han sido utilizadas en el campo de la detección de intrusos son las Redes Neuronales Artificiales y los Agentes Móviles.

El modelado ha sido preciso, eficiente y rápido, ya que, el modelo es el componente esencial en la creación de un nuevo programa. El diseño se ha realizado acorde con las necesidades de seguridad, utilizando un Visual C# 2008 que como se ha descrito anteriormente, es un lenguaje de programación ideal para el presente proyecto.

La investigación aquí planteada será un aporte para el desarrollo del país, ya que es claro que en México los delitos informáticos son cada vez más frecuentes y hace falta mejorar la ley para combatirlos. La Asociación Mexicana de Internet (Amipci) asegura que "La ley federal no establece una definición de lo que es un delito informático". Aunque se contempla como tal el acceso ilícito a una computadora, haría falta una definición más específica. Solucionar las consecuencias de un ataque informático en México es caro, puede oscilar entre 30.000 y 40.000 pesos (de 2.419 a 3.225 dólares), un costo alto para una PYME, la mayor parte del tejido empresarial del país [80].