

# Herramientas para el Análisis de Configuración de Cortafuegos

Claudia G. Reinoso H.  
Universidad Técnica Federico Santa María

Horst von Brand  
Universidad Técnica Federico Santa María

1 de octubre de 2002

## Resumen

Hoy en día, un cortafuegos constituye un elemento obligatorio para la seguridad de una red de computadores. Sin embargo, su utilidad es relativa si la configuración presenta problemas de redundancia o inconsistencia. En general, estos problemas surgen de la dificultad para visualizar el comportamiento del cortafuegos como un todo, el cual está determinado por archivos de texto plano de hasta miles de líneas que establecen las reglas para el control de acceso. Para enfrentar estos problemas, se han desarrollado una serie de mecanismos que los solucionan total o parcialmente. Este documento describe las principales metodologías desarrolladas hasta ahora para enfrentar los problemas de inconsistencia, redundancia e incluso interfaz que se pueden presentar al momento de configurar un cortafuegos. El objetivo, es detectar cuales son las áreas de futuros desarrollos en esta materia. Palabras clave: Cortafuegos, Seguridad.

## 1. Introducción

Un cortafuegos o *firewall* es un componente de la red que ha sido extensamente definido en la literatura. Un ejemplo de ello puede apreciarse en [19] o [27]. En general, un cortafuegos se define como un conjunto de componentes o sistema que se ubica entre dos redes (generalmente la red local e Internet) que tiene las siguientes propiedades: (1) Todo el tráfico desde la red local al exterior y viceversa debe pasar a través de él, (2) solamente el tráfico autorizado, definido en las políticas de seguridad locales, puede atravesar el cortafuegos, (3) el cortafuegos en sí mismo es inmune a la penetración. En cada cortafuegos se define una lista de reglas que establecen cuales paquetes están au-

torizados a atravesarlo y cuales no, lo cual constituye el principal aspecto de la configuración, como puede verse en [13], [20] y [24]. Con tales características podría pensarse que la implementación de un cortafuegos para la protección de la red local representa una buena medida para reforzar la seguridad del sistema. Sin embargo, este tipo de sistema de seguridad presenta una utilidad y eficacia relativa si no se configura adecuadamente. El principal problema que debe enfrentarse cuando se configura un cortafuegos en redes de tamaño mediano, es que las listas de acceso o conjunto de reglas son extensas (cientos de líneas), y en general son escritas en forma de sentencias de un lenguaje de bajo nivel de abstracción, por lo cual es difícil para una persona interpretarlas, administrarlas y mantenerlas. De aquí podemos destacar una serie de problemas:

- Se torna cada vez más difícil determinar cual es el significado del conjunto de reglas, sobre todo cuando deben ser administradas por distintas personas a lo largo del tiempo.
- Es difícil validar el conjunto de reglas contra una política de seguridad, es decir, saber si el conjunto de reglas definidas en el cortafuegos refleja adecuadamente las políticas de seguridad de una organización.
- A veces no se puede determinar con exactitud cual será el efecto de una modificación en el conjunto de reglas, lo que puede tener como consecuencia un problema de inconsistencia o redundancia entre las reglas existentes y las incorporadas recientemente.
- La latencia aumenta en la medida que el conjunto de reglas se hace más extenso. Esto hace deseable establecer una configuración lo menos redundante posible.

Con el objetivo de dar solución a uno o varios de los problemas presentados anteriormente, se han desarrollado una serie de herramientas que implementan distintos algoritmos que exploran una forma alternativa para la representación de las reglas y mecanismos para su análisis. De esta manera se desea proveer a los administradores de red una herramienta que permita entender y analizar la configuración del cortafuegos, puesto que es un componente importante dentro del proceso de administración de la seguridad de una red. Este documento presentará un resumen de las principales herramientas y algoritmos desarrollados hasta ahora para el análisis de las reglas de un cortafuegos y se encuentra organizado de la siguiente forma: En la sección 1, 2 y 3 se describen los principales enfoques existentes en materia de análisis de configuración de un cortafuegos: Diagramas de Decisión Binarios [15], Sistemas Expertos [18] y Simulación [4] respectivamente, para posteriormente clasificarlos de acuerdo a las áreas que cubre en esta materia.

## 2. Diagramas de Decisión Binarios (DDB)

### 2.1. Objetivo

El enfoque desarrollado por Scott Hazelhurst en [15], apunta principalmente a desarrollar una forma alternativa para la **representación del conjunto de reglas o listas de acceso**, la cual permita finalmente obtener una mejor visualización de la configuración, y en consecuencia, un mejor análisis y administración.

### 2.2. Idea General

Se plantea que el conjunto de reglas que determina la configuración del cortafuegos puede ser comprendido como una **expresión booleana**, desde el momento que una regla es una sentencia de la forma **if condición then acción** [1], donde la acción puede ser aceptar o rechazar un determinado paquete. Si se comprende un conjunto de reglas de esta manera, la representación “amigable” de éstas es posible si se encuentra una forma de representación de expresiones booleanas que sea fácilmente interpretable y visualizable. Para lograr este objetivo se utilizan Diagramas de Decisión Bina-

rios [1], un método compacto para representar y manipular expresiones booleanas [7] que se constituye básicamente como un modelo de grafo de una función lógica.

### 2.3. Conceptos Básicos

Como se expresó anteriormente, es posible lograr una mejor representación de las reglas de un cortafuegos si entendemos éstas como expresiones booleanas de la forma **if-then-else**, y utilizamos un método tal como los **Diagramas de Decisión Binarios** para lograr una representación alternativa (y más amigable) de las expresiones booleanas. Esto nos permitiría finalmente obtener una visualización más clara del conjunto de reglas. Sin embargo, para comprender este enfoque, es necesaria una introducción a las expresiones booleanas, el operador **if-then-else** y los DDB.

#### 2.3.1. Expresiones Booleanas

La forma clásica de trabajar con valores de verdad consiste en: las variables booleanas  $x, y, z, \dots$ , las constantes *verdadero* 1 y *falso* 0, y los operadores de *conjunción*  $\wedge$ , *disyunción*  $\vee$ , *negación*  $\neg$ , *implicancia*  $\Rightarrow$  y *doble-implicancia*  $\Leftrightarrow$  los cuales juntos forman las expresiones booleanas. Las prioridades de los operadores son, comenzando con el más alto primero  $\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow$ . a continuación se muestran las tablas de verdad:

	$\neg$		$\wedge$	0	1		$\vee$	0	1		$\Rightarrow$	0	1
0	1		0	0	0		0	0	1		0	1	1
1	0		1	0	1		1	1	1		1	0	1

$\Leftrightarrow$	0	1
0	1	0
1	0	1

#### 2.3.2. El operador if-then-else

Sea  $x \rightarrow y_0, y_1$  el operador *if-then-else* definido por

$$x \rightarrow y_0, y_1 = (x \wedge y_0) \vee (\neg x \wedge y_1)$$

De aquí,  $t \rightarrow t_0, t_1$  es verdadero si  $t$  y  $t_0$  son verdaderos o si  $t$  es falso y  $t_1$  es verdadero.  $t$  se conoce como la expresión de prueba, es decir, el resultado obtenido una vez que se ha evaluado la expresión booleana con alguna instancia en las variables. Es posible expresar todos los operadores booleanos mencionados anteriormente uti-

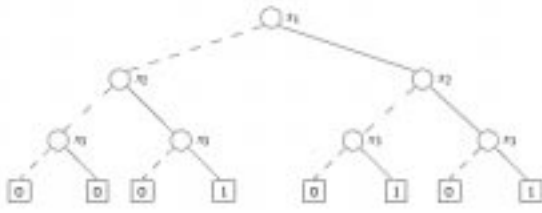


Figura 1: Un DDB simple para  $(x_1 \wedge x_2) \vee x_3$

lizando solamente el operador **if-then-else** y las constantes 0 y 1. Por ejemplo,  $\neg x$  es  $(x \rightarrow 0, 1)$ ,  $x \Leftrightarrow y$  es  $x \rightarrow (y \rightarrow 1, 0), (y \rightarrow 0, 1)$ . Esto nos permite establecer que es posible transformar cualquier expresión booleana en una *Forma Normal if-then-else*, es decir, una expresión booleana construida solamente a partir del operador *if-then-else*. Los detalles de este procedimiento se pueden encontrar en [1].

### 2.3.3. Diagramas de Decisión Binarios

Un Diagrama de Decisión Binario [3] es un método para representar expresiones booleanas. Por ejemplo, la expresión booleana  $(x_1 \wedge x_2) \vee x_3$  puede ser representada por el diagrama de decisión de la figura 1.

Para evaluar una determinada expresión booleana, respecto a un conjunto de valores iniciales asociados a sus variables, debemos comenzar a recorrer el DDB desde la raíz hacia los nodos o terminales. Si la variable tiene el valor 0, se debe elegir el camino de líneas entrecortadas; si la variable tiene el valor 1, se escoge el camino de línea continua. Siguiendo este procedimiento, es posible evaluar la función fácilmente. Adicionalmente a esto, Bryant [6] introduce el concepto de Diagrama de Decisión Binario ordenado y reducido, el cual elimina los terminales, no terminales y sub-expresiones duplicadas o redundantes logrando un DDB abreviado que presenta las siguientes propiedades:

- Son representaciones compactas de expresiones booleanas.
- Para un orden determinado de las variables, la representación de los DDB de una expresión booleana es canónica. Esto significa que, si por ejemplo, construimos el DDB para

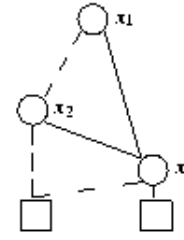


Figura 2: DDB reducido para  $(x_1 \wedge x_2) \vee x_3$

$\neg(a \wedge (b \vee c))$  y  $(\neg a \vee \neg b) \wedge (\neg a \vee \neg c)$  se obtiene exactamente el mismo DDB.

La figura 2 muestra la representación del DDB ordenado y reducido para  $(x_1 \wedge x_2) \vee x_3$ . Como puede apreciarse, el tamaño del diagrama se reduce significativamente.

## 2.4. Implementación

### 2.4.1. Convertir una regla en una expresión booleana

Una vez introducidos los conceptos básicos necesarios para comprender este enfoque, se describirá cómo una regla (o lista de acceso) de un cortafuegos puede ser convertida en una expresión booleana (la cual se representará como un DDB). Debido a que este algoritmo fue diseñado para el análisis de reglas para sistemas CISCO [25], se consideran los siguientes componentes clave:

- Acción: permit, reject.
- Protocolo de paquetes: TCP, UDP, ICMP u otro.
- Dirección de origen del paquete.
- Dirección de destino.
- Puertos.

Cada uno de estos componentes debe representarse en forma de variables booleanas para poder representar la información de la regla en un DDB.

### 2.4.2. Representación de números como vectores de bits

El primer paso para lograr la representación lógica de una regla será representar los números como vectores de bits, es decir, como vectores de DDBs. Esto es debido a que cada bit se entenderá como una variable booleana y cada variable booleana es una expresión representable en un DDB. En consecuencia, la manipulación simbólica de los números puede hacerse como operaciones de bits, siendo posible la implementación eficiente de la suma y la comparación, entre otras. Una de las razones por la que muchas expresiones numéricas pueden ser representadas eficientemente como vectores de bits, es que las sub-expresiones comunes comparten los mismos DDB. Veamos un ejemplo: El segmento de una dirección de red es un número entre 0 y 255. A un nivel más bajo, el segmento de una dirección IP es un vector de 8 bits. Utilizando DDBs, es posible representar un conjunto de números simbólicamente y realizar en ellos muchas operaciones eficientemente. Por ejemplo, para representar el número de 8 bits  $x$  simbólicamente, se utilizará un vector de bits  $\langle x_7, \dots, x_0 \rangle$ , donde cada una de las variables  $x_i$  son variables booleanas (DDBs). Entonces, la condición de que el vector  $x$  es igual a 3, es  $\langle x_7, \dots, x_0 \rangle = \langle f, f, f, f, f, f, t, t \rangle$ . Utilizando la definición de igualdad de vectores, se obtiene la expresión  $x'_7 x'_6 x'_5 x'_4 x'_3 x'_2 x_1 x_0$ . Para hacer la representación de las fórmulas más compacta, la presencia o ausencia de apóstrofe (') indicará el valor 0 o 1 respectivamente y la conjunción ( $\wedge$ ) entre variables del mismo tipo (por ejemplo, las variables que indican el protocolo), se representará con posiciones conjuntas. Por ejemplo, la condición  $x = 2$  se representa por  $x'_7 x'_6 x'_5 x'_4 x'_3 x'_2 x_1 x_0$ , la condición  $x = 2 \vee x = 3$  se representa por  $x'_7 x'_6 x'_5 x'_4 x'_3 x'_2 x_1 x_0 \vee x'_7 x'_6 x'_5 x'_4 x'_3 x'_2 x_1 x'_0$ , lo cual es simplemente  $x'_7 x'_6 x'_5 x'_4 x'_3 x'_2 x_1 x_0$ . Expresiones más grandes pueden ser representadas con DDBs.

### 2.4.3. Variables booleanas para los componentes de una regla

Una vez clara la idea de que representaremos los números como vectores de bits, representaremos toda la información de una regla en forma de variables y expresiones booleanas. Se asignará a los protocolos un número  $0, \dots, n_p - 1$ . Estos números pueden ser representados en  $m_p = \log_2 n_p$  bits, por lo cual se introducen  $m_p$  variables  $\pi_0, \dots, \pi_{m_p-1}$  para codificar el protocolo utilizado. En el ejem-

plo que se muestra a continuación los protocolos pueden ser codificados en 2 bits.

Por cada segmento de la dirección fuente se introducen 8 variables de la forma  $sax[0], \dots, sax[7]$ , donde  $x$  es el número del segmento (recuerde que la dirección IP está constituida por 4 segmentos de 8 bits cada uno). Por cada segmento de la dirección destino, se introducen 8 variables de la forma  $dax[0], \dots, dax[7]$  donde  $x$  es el número del segmento.

Además, pueden especificarse 64000 puertos, los cuales pueden representarse en 16 bits, de esta manera, se introducen 16 variables booleanas ( $p[15], \dots, p[0]$  con  $p[15]$  como bit más significativo, para codificar el número del puerto.

Este enfoque contempla condiciones más complejas, como por ejemplo el rango de puertos y las máscaras de red, sin embargo, está fuera del alcance de este documento.

### 2.4.4. Ejemplo

Suponga que la siguiente regla: ``permit udp host 200.1.19.8 host 200.1.28.2 eq 80'' se debe representar como expresión booleana. Establecemos una terminología particular para representar los protocolos. En este caso,  $tcp = 3$ ,  $udp = 2$ ,  $igmp = 1$ . Por lo tanto, la expresión booleana que representa el protocolo en este caso es  $\pi'_0 \pi'_1 \pi_2$ . La dirección de origen 200.1.19.1 queda representada por

$$\begin{aligned} sa1[7]sa1[6]sa1'[5]sa1'[4]sa1[3]sa1'[2]sa1'[1]sa1'[0] & \wedge \\ sa2'[7]sa2'[6]sa2'[5]sa2'[4]sa2'[3]sa2'[2]sa2'[1]sa2[0] & \wedge \\ sa3'[7]sa3'[6]sa3'[5]sa3[4]sa3'[3]sa3'[2]sa3[1]sa3[0] & \wedge \\ sa4'[7]sa4'[6]sa4'[5]sa4'[4]sa4[3]sa4'[2]sa4'[1]sa4'[0] & \end{aligned}$$

De la misma forma, se encuentra la expresión booleana para la dirección de destino 200.1.28.2

$$\begin{aligned} da1[7]da1[6]da1'[5]da1'[4]da1[3]da1'[2]da1'[1]da1'[0] & \wedge \\ da2'[7]da2'[6]da2'[5]da2'[4]da2'[3]da2'[2]da2'[1]da2[0] & \wedge \\ da3'[7]da3'[6]da3'[5]da3[4]da3[3]da3[2]da3'[1]da3'[0] & \wedge \\ da4'[7]da4'[6]da4'[5]da4'[4]da4'[3]da4'[2]da4[1]da4'[0] & \end{aligned}$$

Por último, el puerto de destino 80 se representa como

$$\begin{aligned} p'[15]p'[14]p'[13]p'[12]p'[11]p'[10]p'[9] \\ p'[8]p'[7]p[6]p'[5]p[4]p'[3]p'[2]p'[1]p'[0] \end{aligned}$$

#### 2.4.5. Conversión del conjunto de reglas a una expresión booleana

Con la metodología enseñada anteriormente, es posible representar una regla como una expresión booleana. A partir de esto, es posible transformar todo el conjunto de reglas en una expresión lógica, teniendo en cuenta las siguientes consideraciones:

- Si el conjunto de reglas está vacío, no se aceptarán paquetes y por lo tanto la expresión booleana correspondiente es **f**.
- Un paquete es aceptado siempre que calce con cualquiera de las reglas de aceptación (*accept*) pertenecientes al conjunto. En este caso, la expresión booleana correspondiente es la disyunción entre las reglas.
- En el caso de que existan reglas de rechazo (*reject*), el paquete es aceptado siempre que no calce con ninguna regla de rechazo y calce con alguna regla de aceptación. Por lo tanto, la expresión booleana correspondiente es la conjunción de la negación de la expresión booleana que representa la regla de rechazo, y el resto de las reglas de aceptación.

Una vez que se encuentra la representación lógica para el conjunto de reglas, se transforman en DDBs con el fin de lograr una representación compacta. Este algoritmo y el algoritmo para la creación de los DDBs desarrollado por Bryant en [8, 16] se encuentran implementados en el *Sistema para Verificación de Hardware Voss* [22], y se realizaron en un lenguaje funcional propio del sistema basado en ML [10], denominado FL [22].

### 2.5. Representación y Análisis de la Información

Una vez que el DDB que representa el conjunto de reglas ha sido construido, puede ser utilizado para acciones de búsqueda. El simple hecho de recorrer el grafo determina el valor de salida de la expresión mediante el examen secuencial de los valores de los datos de entrada, donde cada nodo del grafo representa una variable y las ramas que salen del nodo representan los posibles valores para esa variable. Cuando un paquete pasa por el cortafuegos, se extrae la información relevante de la cabecera o *header* del paquete. Con esta información es posible recorrer el DDB y determinar:

- La aceptación o rechazo de un determinado paquete.
- Consultas tales como: en qué puertos de la dirección destino *y* se aceptan conexiones de la dirección origen *x*; qué puertos se encuentran accesibles desde el exterior para la dirección *z*, etc.

Por otro lado, la herramienta permite representar la información del conjunto de reglas en forma de tabla, lo cual permitiría una visualización más clara de la configuración del cortafuegos. A continuación se muestra la representación de dos reglas. La línea que comienza con dos puntos muestra los parámetros de búsqueda dados por el usuario, donde *sc* indica el orden de las columnas a desplegar y el parámetro *cond* indica alguna condición particular para el despliegue de la información. En el ejemplo, se muestra la información de las reglas según el protocolo (*proto*) y el puerto (*port*). El tamaño de la tabla dependerá bastante del orden de las columnas. Por ejemplo, si se muestra la información según los puertos, el largo de la tabla será menor que si se muestra la información según las direcciones de origen o destino.

```
: sc[proto,Port] cond;
```

Proto	Ports	Src1	Src2	Src3	Src4
1	0--65535	0--255	0--255	0--255	0--255

Dest1	Dest2	Dest3	Dest4
0--255	0--255	0--255	0--255

El hecho de trabajar con expresiones lógicas, también permite implementar soluciones para visualizar los cambios en el conjunto de reglas, verificar reglas redundantes (si se repiten, o si los valores de la máscara de red cubren los valores en otras reglas, etc). Además, el autor plantea que se pueden implementar otras formas de validación como por ejemplo, el efecto de una regla de rechazo sobre otras reglas de aceptación (estudio de inconsistencia por ejemplo), etc.

## 3. Sistemas Expertos

### 3.1. Objetivo

El enfoque desarrollado por Pasi Eronen y Jukka Zitting en [18] tiene por objetivo implementar

una herramienta que permita realizar consultas interactivas sobre la configuración de un cortafuegos, agregar nuevas reglas de una manera más fácil y desarrollar nuevas funcionalidades que permitan detectar errores comunes de configuración.

## 3.2. Idea General

Este enfoque plantea que el análisis de un cortafuegos debe hacerse desde el punto de vista del flujo de datos de red (básicamente un paquete de datos) y las restricciones que impone a la circulación de éste el conjunto de reglas. Para ello, se plantea que un paquete es un conjunto de variables, donde las variables representan los campos del paquete relevantes al momento de hacer el filtrado. Cada variable tendrá un espacio de valores posibles dependiendo de su tipo (protocolo, puerto, dirección ip, etc.). Cada una de las reglas, representa una restricción dentro del espacio de valores posibles que pueden tomar estas variables. Entonces el problema de saber qué paquetes pueden o no atravesar la red, puede ser resuelto con algoritmos combinatoriales que entreguen los posibles valores de las variables dadas todas las restricciones.

Para llevar a cabo la implementación de esta solución se creó un *Sistema Experto*<sup>1</sup> basado en *Eclipse* [17], un lenguaje de programación lógica [9] especialmente diseñado para resolver problemas combinatoriales y de cumplimiento de restricciones.

## 3.3. Conceptos Básicos

### 3.3.1. Sistema Experto

El objetivo de un sistema experto [12], que consiste en solucionar problemas y responder preguntas, generalmente se alcanza mediante la combinación de un **motor de inferencia lógica** con un **conocimiento base**. La información en el conocimiento base contiene una serie de hechos conocidos y un conjunto de reglas de la forma *si-entonces* las cuales representan el conocimiento existente sobre el dominio de un problema. El motor de inferencia lógica es la unidad de procesamiento que resuelve los problemas presentados realizando inferencias lógicas en los hechos dados y las reglas almacenadas en el conocimiento base.

<sup>1</sup>programa computacional utilizado para resolver problemas y responder preguntas en un dominio de problema que comúnmente requiere de experticia humana

### 3.3.2. Programación Lógica para la Satisfacción de Restricciones (CLP)

La programación lógica [9] utiliza inferencia lógica para resolver problemas. En vez de entregar los pasos computacionales requeridos para resolver el problema, el programa lógico entrega los hechos lógicos y las dependencias que describen una solución y utiliza un motor de inferencia para resolver el problema. Programación lógica con satisfacción de restricciones significa extender las capacidades de la programación lógica para satisfacer restricciones.

## 3.4. Implementación

### 3.4.1. Conocimiento Base

El conocimiento base para este sistema es una colección de reglas lógicas y hechos, expresados en un lenguaje de programación basado en prolog llamado *Eclipse* [17]. En *Eclipse* cada decisión se modela como una variable, y cada elección como un posible valor para esa variable. Por otro lado, las restricciones se modelan como relaciones entre las variables en forma de *predicados* (funciones propias de *Eclipse*) las cuales pueden representar una relación explícita (como un conjunto de hechos) o bien una relación implícita (un conjunto de reglas).

### 3.4.2. Motor de Inferencia

El motor de inferencia también corresponde a *Eclipse*. El cual se encarga de resolver el problema combinatorial producto de alguna consulta.

### 3.4.3. Ejemplo

Para comprender la manera de implementar este enfoque presentaremos un ejemplo simple. Supongamos que el cortafuegos tiene como única regla ```access-list 100 permit tcp 192.168.1.0 0.0.0.255 host 10.0.0.1 eq 23``` en la sintaxis de sistemas CISCO. El conocimiento base en este caso estará constituido por el predicado que representa a un paquete (flujo de datos) con sus correspondientes variables y la regla en cuestión. El siguiente predicado define

el concepto básico de paquete en el conocimiento base.

```
paquete(protocolo,origen,destino,puerto_origen,
         puerto_destino,flags)←
0 ≤ protocolo ≤ 255 ∧
0 ≤ origen ≤ 4294967295 ∧
0 ≤ destino ≤ 4294967295 ∧
0 ≤ puerto_origen ≤ 65535 ∧
0 ≤ puerto_destino ≤ 65535 ∧
0 ≤ flags ≤ 1
```

Por otro lado, en el sistema se representan las reglas de las listas de acceso como restricciones en el espacio de las variables pertenecientes al predicado *paquete*. Cada regla se asocia con una parte del espacio del paquete (una séxtupla de rangos). El detalle de este enfoque es que se vuelve ineficiente cuando las reglas son traspasadas directamente a un predicado, por ende, deben ser sometidas previamente a una *de-correlación* [21], un proceso que transforma el conjunto de reglas en reglas que no se traslapan (para estos efectos podría estudiarse el uso de [2]). Una vez hecho esto, el conjunto de reglas resultante se agrega al conocimiento base en forma de *match\_list*, a continuación se muestra el predicado correspondiente a la regla de nuestro ejemplo.

```
match_list(100,permit,(protocolo,origen,
                     destino,puerto_origen,puerto_destino,
                     flags)) ←
protocolo=6 ∧
3232235776 ≤ origen ≤ 3232236031 ∧
destino = 167772161 ∧
0 ≤ puerto_origen ≤ 65535 ∧
puerto_destino = 23 ∧
≤ flags ≤
```

Basándose en estos predicados, el motor de inferencia lógica es capaz de encontrar respuesta a preguntas tales como “¿Cuál es la acción para este paquete?”, “¿Qué paquetes son permitidos por esta lista de acceso?”.

#### 3.4.4. Desarrollo de Nuevas Funcionalidades

Es posible incrementar la capacidad para responder preguntas más complejas agregando información auxiliar en forma de nuevas reglas o nuevos hechos al conocimiento base, en forma de predicados de mayor nivel, tales como la topología de la red, o las características de conexiones específicas, etc.

### 3.5. Representación y Análisis de la Información

El análisis de la información queda completamente delegado al motor de inferencia lógica que provee *Eclipse*. Este se encarga de resolver el problema combinatorial que se genera después de una consulta. El sistema puede responder consultas más o menos complejas dependiendo de la información existente en el conocimiento base, pero básicamente la herramienta implementa los siguientes tipos de consultas. (1) Consultas y operaciones sobre las listas de acceso en sí mismas. (2) Consultas sobre los flujos de datos permitidos por las listas de acceso. (3) Reglas expertas o de alto nivel para reconocer y resolver problemas comunes de configuración.

La información se presenta en forma de documento de texto, utilizando las bibliotecas de entrada y salida (I/O) de *Eclipse* y el lenguaje de programación Prolog [26]. Si por ejemplo se quiere preguntar qué servicios están accesibles desde un determinado *host*, es posible responder utilizando únicamente los predicados *match\_list* vistos en la sección 3.4.3. En cuanto se realiza la consulta, el motor de inferencia lógica consultará al conocimiento base para encontrar todas las posibles sustituciones para la(s) variable(s) libre(s) en una determinada condición. Los componentes de la interfaz de usuario recolectan y presentan los resultados en un formato legible por el usuario. A continuación se muestra cómo funciona la herramienta para una consulta que solicita mostrar todos los servicios en el *host* 10.10.11.8.

**? show\_services on 10.10.11.8**

```
UDP services on 10.10.11.8
port from
any 10.10.10.0/24
```

```
TCP services on 10.10.11.8
port from
<=79 10.10.10.0/24
80 0.0.0.0 - 10.10.10.255
0.0.0.0 - 10.10.10.255
10.10.11.128 - 255.255.255.255
>=81 10.10.10.0/24
```

El reporte no muestra solamente los servicios permitidos, sino que también muestra las direcciones origen que pueden utilizar esos puertos. Podemos ver que el puerto 80 es accesible desde cualquier sitio, pero otros puertos solamente pueden ser accedidos desde IPs pertenecientes a la red interna.

## 4. Simulación

### 4.1. Objetivo

El enfoque propuesto por Mayer en [4] y [5], tiene por objetivo diseñar una herramienta que permita configurar un cortafuegos **en conformidad con las políticas de seguridad de la organización**, abstrayéndose de la topología específica de la red y de la complejidad de los archivos de configuración, generándolos automáticamente. Además, provee una interfaz que ilustra las reglas en una manera gráfica y permite analizarlas mediante consultas interactivas. Con esta representación, es más fácil para el administrador detectar errores de configuración.

### 4.2. Idea General

La principal característica de la herramienta desarrollada en los laboratorios Bell, es que modela la red como un conjunto de objetos conectados entre sí física y lógicamente. La conexión física está determinada por la topología propia de la red, mientras que las conexiones lógicas están dadas por los flujos de datos permitidos por las políticas de seguridad. Cada objeto presenta atributos como IP, nombre, servicios que provee, permisos de acceso, etc., los cuales dependerán de la participación o rol que cada objeto tiene dentro de la red. La forma o metodología para modelar una red y las políticas de seguridad está definida por un **modelo entidad-relación**. Este modelo define un *framework* o marco de trabajo basado en roles para modelar las políticas de seguridad, lo cual permite abstraerse de la topología de red al momento de definirlas. Cada rol establece las propiedades que puede asumir cada elemento de la red (servicios, protocolos, puertos permitidos para el *host*). La implementación o instanciación del modelo se logra utilizando un lenguaje propietario denominado MDL (**Model Definition Language**), un lenguaje orientado a objetos similar a C++, donde cada objeto es presentado como una estructura de una serie de variables determinadas por el rol del objeto. Una vez que se ha modelado la topología de la red y las políticas de seguridad, se utiliza un **compilador** que se encarga de transformar la instancia del modelo a los archivos específicos de configuración del cortafuegos. Para el análisis de la configuración del cortafuegos se implementa un **ilustrador de reglas**, el cual se encarga de repre-

sentar gráficamente el conjunto de reglas en forma de políticas de seguridad, y un **motor de consultas** interactivas, una combinación de algoritmos para el manejo de grafos y un simulador que frente a una consulta simula el comportamiento de los paquetes descritos por la consulta en la medida que estos atraviesan la red (previamente modelada). Con esto es posible evaluar la correctitud de las políticas de seguridad implementadas. Este enfoque es muy similar en esencia al desarrollado por Guttman en [14]. Como el enfoque de Mayer es complementario, nos centraremos en éste y se deja el trabajo de Guttman a inquietud del lector.

### 4.3. Conceptos Básicos

#### 4.3.1. Modelo Entidad-Relación

Como se mencionó anteriormente, la principal característica de este enfoque es que modela la red como un conjunto de objetos relacionados entre sí. De esto se desprende el hecho de que es necesaria la incorporación de información adicional a las reglas del cortafuegos, tal como la información de la topología de la red y las conexiones. Para asignar atributos a los objetos de la red, se introduce el concepto de *rol*. Un rol es una propiedad que puede ser asumida por diferentes *hosts* en la red. Los roles definen la capacidad de iniciar y aceptar servicios. Por ejemplo, un rol `servidor_correo` debería definir la capacidad de aceptar el servicio de correo (smtp, por ejemplo, tcp en el puerto 25) desde cualquier lugar. De este modo, una política de seguridad ahora podría definir los permisos sobre el rol `servidor_correo` y no sobre la máquina específica. La topología de la red también se modela a través de un enfoque entidad-relación. La entidad *host* modela una máquina en la red con su propia dirección IP y nombre. Los roles son asignados a un *host*, o a un grupo de *hosts*

#### 4.3.2. MDL Model Definition Language

Este lenguaje de programación fue desarrollado con el objetivo de permitir la instanciación de las políticas de seguridad y posteriormente traducir las políticas definidas en base a roles, en una política basada en la topología de la red. Cada vez que se desea introducir una política de seguridad, debe programarse en MDL, así como también la topología de la red en primera instancia. Además,



se implementa un *parser* para MDL, el cual transforma el programa en MDL en una instancia del modelo entidad-relación. El modelo es expresado por la estructura de datos correspondiente en C++.

#### 4.3.3. Compilador

Después de que el administrador ha diseñado y programado las políticas de seguridad en MDL y el *parser* ha generado el modelo entidad-relación, es necesario trasladar este modelo a los archivos de configuración propios de firewall. La traducción debe garantizar que las reglas resultantes implementan correctamente la política de seguridad diseñada.

#### 4.3.4. Ilustrador de Reglas

El ilustrador de reglas traduce el conjunto de reglas del firewall en representaciones basadas en grafos que visualizan la estructura de los grupos de *hosts* y los servicios que el cortafuegos permite.

#### 4.3.5. Motor de Consultas

Cada vez que se realiza una consulta comienza a funcionar un motor de búsqueda. Este motor es una combinación de algoritmos para el manejo de grafos y un simulador. Cada vez que el usuario desea analizar el comportamiento del cortafuegos frente a ciertos flujos de datos y se realiza la consulta respectiva, el sistema simula el comportamiento de los paquetes descritos por la consulta en la medida que estos atraviesan la red modelada. Cada vez que se visita un nodo, se analiza si las políticas de seguridad (ahora plasmadas en forma de roles) permiten el paso de los paquetes para ese host. Eventualmente, las reglas permiten pasar solamente a un subconjunto de la consulta inicial. En este caso, se subdivide la consulta sucesivamente en la medida que atraviesa la red hasta llegar a el grupo de *hosts* destino. Los resultados se determinan por los paquetes simulados que han sido capaces de llegar a destino.

### 4.4. Implementación

La parte clave es la utilización del lenguaje de programación MDL. A través de éste se elaboran todos los roles, *hosts*, grupos de *hosts* y roles que

poseen. Una vez hecho esto, todo el trabajo queda por parte de la herramienta.

### 4.5. Análisis y Representación de la Información

La representación del conjunto de reglas se hace en forma gráfica, utilizando el ilustrador de reglas, o bien puede mostrarse en forma de texto cuando se trata de responder a consultas específicas acerca del funcionamiento del cortafuegos. Para la interfaz de consulta respuesta, se tiene una interfaz de usuario gráfica, desarrollada utilizando Qt [11] y C++ [23]. La detección de errores de configuración comunes, tales como redundancia e inconsistencia, pueden detectarse al poseer una visualización más amigable, pero no se implementa ningún mecanismo explícito para ello.

## 5. Clasificación

El objetivo de este documento es mostrar el avance existente en materia de análisis de configuración de cortafuegos. Para tener una visión más clara de lo desarrollado hasta ahora, se ha establecido una serie de características que se consideran de mayor importancia al momento de analizar la configuración de un cortafuegos:

- **Conformidad con las Políticas de Seguridad:** Se refiere a que el método permite establecer que las reglas de un cortafuegos se encuentran en concordancia con las políticas de seguridad de la organización. Esta característica se puede encontrar en forma **explícita** o **implícita**. Se encuentra en forma explícita cuando existe un mecanismo encargado de validar la conformidad entre las reglas del cortafuegos y las políticas de seguridad. Se encuentra implícita cuando la verificación se deriva de la observación y análisis de otras facilidades que provee la herramienta en cuestión.
- **Correctitud:** Se refiere a que el método permite detectar y/o corregir errores en las listas de acceso, ya sea errores de redundancia, inconsistencia, y errores típicos de configuración. Esta característica también se puede encontrar en forma **explícita** o **implícita**. Se encuentra en forma explícita cuando existe un

mecanismo encargado de detectar y/o corregir los errores existentes en la configuración. Se encuentra implícita cuando la detección de errores se deriva de la observación y análisis de otras facilidades que provee la herramienta.

El siguiente cuadro muestra la clasificación de las distintas herramientas estudiadas.

	Conformidad	Correctitud
<b>DDB</b>	Implícito	Explícito
<b>S. Experto</b>	Implícito	Explícito
<b>Simulación</b>	Explícito	Implícito

## 6. Conclusiones

Este documento presentó una descripción no muy extensa de los métodos para el análisis de la configuración de un cortafuegos existentes en la literatura, lo cual permitirá al lector tener una idea general del desarrollo científico existente en esta área. Aunque podría pensarse que el análisis de la configuración de un cortafuegos es una tarea específica y acotada, tener una visión general de los trabajos realizados hasta ahora ayuda a determinar distintas áreas, cada una diferente pero complementaria a la otra, en las que es necesario un desarrollo. Entre ellas podemos destacar:

- Diseño de una interfaz que permita realizar consultas en forma amigable, lo que constituye el punto más débil de la mayoría de los enfoques estudiados.
- Dentro de la misma área, es necesario desarrollar mejores representaciones gráficas, las cuales permitan la visualización de las relaciones entre los elementos de una red y las restricciones de flujos entre los distintos elementos.
- Determinación de los algoritmos más eficientes en términos de desempeño y eficacia en la detección de inconsistencia y redundancia, ya que los algoritmos utilizados son muy distintos en metodologías y este sería un buen parámetro de comparación.
- Integración de los aspectos destacables de cada enfoque.

Por otro lado, es posible destacar que no existen trabajos orientados a mejorar la forma en que están constituidos los archivos de configuración de

un cortafuegos (en forma de listas). Quizás deberían explorarse nuevas formas de representación que ayudaran a dificultar la aparición de errores y a facilitar la representación de las listas en forma de políticas de seguridad. Los trabajos futuros estarán orientados a la investigación en esa área.

## Referencias

- [1] Henrik Reif A. *An Introduction to Binary Decision Diagrams*, 1997.
- [2] Subhash Suri y Guru Palukar Adishesu Hai. Detecting and resolving packet filter conflicts. *IEEE INFOCOM 2000 - The Conference on Computer Communications*, no. 1, pp. 1203 - 1212, Marz. 2000.
- [3] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, vol. c-27, no. 6, Junio 1978.
- [4] Elisha Ziskind Alain Mayer, Avishai Wood. Fang: A firewall analysis engine. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*.
- [5] Yair Bartal, Alain J. Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31, 1999.
- [6] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, September 1992.
- [7] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, vol.24, no.3, Sept. 1992.
- [8] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, vol. c-35, no.8, Agosto 1986.
- [9] Jackes Cohen. Constraint logic programming. *Communications of the ACM*, 33(7):52-58, Jul. 1990.
- [10] Andrew Cumming. *A Gentle Introduction to ML*, 1998.
- [11] M.K. Dalheimer. *Programming With Qt*. O'Reilly and Associates, Inc., 1999.

- [12] Robert S. Engelmores and Edward Feigenbaum. Expert systems and artificial intelligence. In Knowledge-based systems in Japan. Japanese Technology Evaluation Center.
- [13] Mark Grennan. *Firewall and Proxy Server HOWTO v0.80*, Feb. 2000.
- [14] Joshua Guttman. Filtering postures: Local enforcement for global policies. In *Proc. IEEE Symp. on Security and Privacy, Oakland, CA*, 1997.
- [15] Scott Hazelhurst. Algorithms for analysing firewall and router access lists. *Technical Report TR-Wits-CS-1999-5*, 1999. Shortened version appeared in Workshop on Dependable IP Systems and Platforms, In Proc ICDSN, June 2000.
- [16] R.L. Rudell K.S. Brace and R.E. Bryant. Efficient implementation of a bdd package. *Proc. 27th Design Automation Conference*, 1990.
- [17] Mark Wallace Stefano Novello and Joachim Schimpf. Eclipse: A platform for constraint logic programming., 1997. Imperial College, London.
- [18] Jukka Zitting Pasi Eronen. An expert system for analyzing firewall rules. *Technical Report IMN-TR-2001-14, In Proceedings of the 6th Nordic Workshop on Secure IT systems*.
- [19] Marcus J. Ranum. A network firewall. In *Proceedings of the First World Conference on System Administration and Security*, 5401 Westbard Ave. Suite 1501, Bethesda, MD 20816, 1992. SANS Institute.
- [20] Rusty Russell. *Linux iptables HOWTO, v0.0.2*, Sept. 1999.
- [21] L. A. Sanchez. Security policy protocol, appendix c, 2001. Internet Draft, draft-ietf-ipspp-00.txt.
- [22] Carl-Johan H. Seger. Voss - a formal hardware verification system, user's guide. Technical report, Department of Computer Science, University of British Columbia, 1993.
- [23] Bjarne Stroustrup. *The C++ Programming Language (Special 3rd Edition)*.
- [24] Cisco Systems. *Basic Firewall Configuration*. [http://www.cisco.com/warp/public/110/top\\_issues/pix/pix\\_index.shtml](http://www.cisco.com/warp/public/110/top_issues/pix/pix_index.shtml).
- [25] Cisco Systems. *Configuring Access Control Lists*. [http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sft\\_6\\_1/configgd/acc\\_list.htm](http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sft_6_1/configgd/acc_list.htm).
- [26] C. S. Mellish W. F. Clocksin. *Programming in Prolog*. Springer Verlag, 1994.
- [27] Steven M. Bellovin William R. Cheswick. *Firewalls and Internet Security : Repelling the Wily Hacker*. Addison-Wesley, 1994.