Análisis de anomalías sobre políticas de control de acceso en red

Joaquín García-Alfaro[1,2], Frédéric Cuppens[1], Nora Cuppens-Boulahia[1]

[1] Département Réseaux, Sécurité et Multimédia,
Ecole Nationale Supérieure des Télécommunications de Bretagne,
02, rue de la Châtaigneraie, 35576 Cesson Sévigné, Rennes - Francia
Email: {frederic.cuppens,nora.cuppens}@enst-bretagne.fr

[2] Departament d'Enginyeria de la Informació i les Comunicacions,
ETSE, Universitat Autònoma de Barcelona,
Edifici Q, 08193 Bellaterra, Barcelona - España
Email: joaquin.garcia@deic.uab.es

Resumen La mayor parte de compañías y organizaciones actuales utilizan sistemas o dispositivos cortafuegos para controlar el acceso a su infraestructura de red. Para ello, es necesario configurar dichos sistemas con un conjunto de reglas de filtrado. Sin embargo, la existencia de errores o anomalías en estas reglas puede degradar la política de seguridad de la red. Descubrir y eliminar estos errores de configuración es un problema serio y complicado de solucionar. En este artículo, presentamos una serie de algoritmos para la gestión de esta problemática. Nuestro enfoque está basado en el análisis de relaciones entre el conjunto de reglas, así como un proceso de transformación y exclusión de información, de forma que la configuración del sistema cortafuegos quede completamente libre de anomalías. A su vez, nuestros algoritmos detectan y proporcionan al administrador del sistema el motivo por el cual dichas reglas deben ser modificadas (o eliminadas) de la configuración inicial.

Palabras clave: Seguridad en Redes, Control de Acceso, Sistemas Cortafuegos

1. Introducción

El uso de sistemas cortafuegos es el método más utilizado por compañías y organizaciones para segmentar el control de acceso y vigilar el tráfico de sus redes. Generalmente son empleados para realizar un filtrado del tráfico que viaja entre zonas de confianza e Internet, tanto para supervisar su interacción de entrada como de salida.

Los sistemas cortafuegos son componentes tradicionales de seguridad perimetral, con distintas interfaces por las que controlar el tráfico de la red. Una compañía

puede dividir su red en, por ejemplo, tres zonas diferentes: una zona desmilitarizada, un red privada y una zona para la administración general. En este caso, el sistema cortafuegos será configurado con tres interfaces, una para cada zona, más una cuarta para poder controlar el acceso de entrada, o de salida, a Internet.

Para poder aplicar correctamente el proceso de filtrado, es necesario configurar el sistema cortafuegos con un conjunto de reglas de filtrado (como, por ejemplo, el conjunto de reglas mostrado en la tabla 1). Cada regla especifica, por norma general, una *decisión* (por ejemplo, *aceptar* o *denegar*) que será aplicada a un conjunto de *condiciones*, como el protocolo, el origen, el destino, etcétera. Para nuestro trabajo, definimos una regla de filtrado de la manera siguiente:

$$R_i : \{condition_i\} \rightarrow decision_i \tag{1}$$

donde i es la posición relativa de la regla dentro del conjunto de reglas y $decision_i$ es un valor booleano en $\{accept, deny\}$. Por otro lado, $\{condition_i\}$ es una expresión lógica en forma normal conjuntiva, igual a $A_1 \wedge A_2 \wedge ... \wedge A_p$, siendo p el número de atributos de dicha expresión.

En el siguiente ejemplo se muestra el conjunto de reglas de filtrado de la tabla 1 utilizando el formalismo anterior[3].

$$\begin{aligned}
&R_1 : (s \in [10, 50] \wedge d \in [20, 60]) \rightarrow deny \\
&R_2 : (s \in [40, 90] \wedge d \in [50, 40]) \rightarrow accept \\
&R_3 : (s \in [60, 100] \wedge d \in [20, 80]) \rightarrow accept \\
&R_4 : (s \in [30, 80] \wedge d \in [30, 100]) \rightarrow deny \\
&R_5 : (s \in [1, 70] \wedge d \in [20, 90]) \rightarrow accept
\end{aligned}$$

Para evitar conflictos de solapamiento durante el procesado de paquetes, la mayoría de sistemas cortafuegos implementa una estrategia de ordenación de reglas (o *first matching*) para que, en el caso de que un mismo paquete sea asociado a más de una regla, sea tan solo la regla con mayor prioridad en el orden la que finalmente acabe siendo aplicada. Esta estrategia introduce, sin embargo, nuevos conflictos en la configuración del cortafuegos como, por ejemplo, redundancia o enmascaramiento (*shadowing*) de reglas. Definimos estos dos casos de conflicto de la manera siguiente.

[3] Para simplificar, el número de atributos para cada condición (p) es igual a dos: (s)ource y (d)estination. No se mostrarán en el ejemplo el resto de atributos (p)rotocol, (sP)ort y (dP)ort, ya que su valor será siempre igual a $true$.

orden	condición					decisión
	(p)rotocol	(s)ource	(sP)ort	(d)estination	(dP)ort	
1	any	xxx.xxx.xxx.[010,050]	any	xxx.xxx.xxx.[020,060]	any	deny
2	any	xxx.xxx.xxx.[040,090]	any	xxx.xxx.xxx.[050,040]	any	accept
3	any	xxx.xxx.xxx.[060,100]	any	xxx.xxx.xxx.[020,080]	any	accept
4	any	xxx.xxx.xxx.[030,080]	any	xxx.xxx.xxx.[030,100]	any	deny
5	any	xxx.xxx.xxx.[001,070]	any	xxx.xxx.xxx.[020,090]	any	accept

Cuadro 1. Ejemplo de un conjunto de reglas de filtrado.

Definición 1 *Sea R un conjunto de reglas de filtrado. Decimos que una regla R_i dentro de R es enmascarada (**shadowed**) si y solo si todo el tráfico susceptible de ser asociado con R_i es previamente asociado por, al menos, una regla o combinación de reglas con mayor prioridad en el conjunto R. Por este motivo, podemos afirmar que la regla R_i dentro del conjunto R no será nunca utilizada.*

Definición 2 *Sea R un conjunto de reglas de filtrado. Decimos que dentro de R se produce **redundancia** si y solo si existe, al menos, una regla R_i dentro de R, para la cual se cumplen las siguientes condiciones: (1) R_i no es enmascarada por ninguna otra regla; (2) en el caso de eliminar R_i de R, la política de filtrado del cortafuegos no se ve alterada.*

La mayor parte de la investigación actual sobre sistemas cortafuegos se centra en el estudio de nuevos diseños o arquitecturas [12,10,3], en la mejora de sus procesos de detección [14,11,8], así como de sus mecanismos de clasificación de paquetes [16,7,15]. Sin embargo, ninguna de las propuestas anteriores se centra en el estudio de una gestión acorde a los conflictos de configuración enunciados anteriormente.

La detección y eliminación de estos conflictos (redundancia y enmascaramiento) sobre políticas de control de acceso es una tarea compleja pero necesaria, ya que de no ser solucionada apropiadamente, puede conducir al sistema cortafuegos a realizar acciones equivocadas, y provocar un debilitamiento de la política de seguridad de la empresa u organización. Aunque la gestión de estos conflictos está empezando a ser tratada por distintos trabajos como, por ejemplo, [1,9,2,13,5], la mayor parte de resultados y aportaciones son todavía muy limitados.

En este artículo, presentamos una extensión y mejora de los algoritmos para el descubrimiento y eliminación de redundancia y enmascaramiento de reglas de filtrado, presentados en [6].

Nuestro objetivo principal es el siguiente. Dada una configuración de control de acceso inicial, nuestro conjunto de algoritmos realizará un análisis de la configuración proporcionada, para verificar la no existencia de anomalías. En el supuesto de encontrar errores en la configuración, el conjunto de reglas afectado será reescrito, para dar como resultado un conjunto de reglas equivalente, pero libre de conflictos. El principio básico de nuestra propuesta es la búsqueda de relaciones existentes entre condiciones de las distintas reglas de la política proporcionada, tales como coincidencia, disyunción e inclusión. Mediante un proceso de transformación, la política inicial será derivada en un conjunto de reglas equivalente, donde las distintas reglas que lo conforman son completamente libres de cualquier relación posible.

Las ventajas de nuestra propuesta son varias. En primer lugar, después de la reescritura de la política de control de acceso inicial, es posible verificar que no hay redundancia ni enmascaramiento en el conjunto de reglas resultante. Cada regla considerada como innecesaria durante el proceso de análisis será eliminada de la política inicial. En segundo lugar, cuando dicha detección se produzca, nuestro proceso de análisis proveerá una prueba del conflicto sobre la consola de administración del proceso. De esta manera, el administrador en cargo del proceso de auditoría podrá verificar la corrección del evento.

Por último, las reglas de la política de control de acceso resultante serán completamente disjuntas, es decir, la ordenación de reglas del conjunto resultante no tendrá ninguna importancia. De este modo, es posible llevar a cabo una segunda transformación del conjunto de reglas, para generar una política de control de acceso abierta o cerrada: abierta, cuando las reglas apunten únicamente a prohibiciones; cerrada, cuando las reglas apunten únicamente a paquetes permitidos.

Una vez realizada esta segunda reescritura de las reglas de filtrado, el administrador encargado de la política de control de acceso del cortafuegos tendrá una visión más nítida del tráfico rechazado (en el caso de una política abierta) o del tráfico aceptado (en el caso de una política cerrada).

El resto de este artículo está organizado de la siguiente manera. En la sección 2 se presenta un análisis sobre algunos trabajos relacionados con nuestra propuesta. En la sección 3 se presentan nuestros algoritmos, así como una serie de teoremas y demostraciones para dar validez a nuestra propuesta. La sección 4 analiza la complejidad de nuestros algoritmos. También se presenta en esta sección la evaluación realizada sobre una primera implementación de nuestra propuesta. La sección 5 cierra, por último, nuestro artículo, presentando brevemente algunas líneas que podrían dar continuidad al presente trabajo.

2. Antecedentes

Un primer enfoque que trata el problema de los conflictos en sistemas cortafuegos es la aplicación de un modelo formal para la expresión de políticas de seguridad en red. En [5], por ejemplo, se propone un modelo formal basado en OrBAC (*Organizational Based Access Control*) con este propósito. A partir de este lenguaje formal, será generado de forma automática un conjunto de reglas específico para cada sistemas cortafuegos, a través de un proceso de traducción.

Desafortunadamente, este enfoque no es lo suficientemente completo como para asegurar que no se producen futuros errores o anomalías en la configuración final. Es posible que, durante las tareas de administración y mantenimiento del dispositivo cortafuegos, sea modificada la configuración inicial, libre de errores, y se introduzcan errores no controlados por la política formal.

Otras propuestas, como [1,2,9], tratan de dar solución al problema de las anomalías, analizando directamente las reglas de filtrado de cada dispositivo.

En [1], por ejemplo, encontramos un primer trabajo que propone la realización de un proceso de auditoría sobre el conjunto inicial de reglas de filtrado. Sus autores definen un único caso de anomalía, donde dos reglas entran en conflicto cuando se cumplen las siguientes dos condiciones: (1) la primera regla, prioritaria en el orden, es capaz de ser aplicada sobre un conjunto de paquetes asociados a la segunda regla; (2) la segunda regla también puede ser asociada con algunos de los paquetes que se aplican con la primera regla.

A nuestro modo de ver, este primer enfoque es muy limitado, ya que no detecta conflictos completos, como redundancia o enmascaramiento, sino que detecta tan sólo un caso específico de solapamiento entre reglas. Este tipo de conflicto queda mejor definido como una combinación de redundancia y enmascaramiento entre reglas, tal y como definíamos en la sección anterior.

En [9], dos nuevos casos de conflicto entre reglas son tratados. Primero, una regla R_j es definida como *backward redundant* ssi existe otra regla R_i con mayor prioridad en el orden, tal que todos los paquetes que se pueden asociar a R_j son también asociados con R_i.

Por otro lado, una regla R_i es definida como *forward redundant* ssi existe otra regla R_j con la misma decisión, pero menos prioritaria en el orden, y se cumplen las siguientes condiciones: (1) todo paquete que puede ser asociado con la regla R_i, puede ser también asociado con la regla R_j; (2) sea R_k cualquier regla entre R_i y R_j, tal que R_k puede ser asociada a todos los paquetes que coinciden con R_i, R_k tiene la misma decisión que R_i.

Aunque esta propuesta apunta en la buena dirección, consideramos nuestra definición de anomalías más completa, pues todas las reglas *backward* o *forward redundant* son casos específicos de nuestra definición de redundancia y enmascaramiento, pero no viceversa. Por ejemplo, teniendo en cuenta el siguiente conjunto de reglas:

$$R_1 : s \in [10, 50] \rightarrow deny$$
$$R_2 : s \in [40, 70] \rightarrow accept$$
$$R_3 : s \in [50, 80] \rightarrow accept$$

su propuesta de detección, tal y como se ha definido más arriba, no podrá detectar la redundancia de R_2. Consideramos, por tanto, esta propuesta como incompleta.

Por último, en [2] encontramos un conjunto de técnicas y algoritmos para la detección de redundancia y enmascaramiento. Sin embargo, y aunque la eficiencia de esta última propuesta parece bastante prometedora, consideramos también este nuevo enfoque como incompleto.

En primer lugar, su propuesta es demasiado débil ya que, analizando sus algoritmos de detección, es sencillo encontrar posibles anomalías que no serán detectadas. Por ejemplo, suponiendo las siguientes reglas de filtrado:

$$R_1 : s \in [10, 50] \rightarrow accept$$
$$R_2 : s \in [40, 90] \rightarrow accept$$
$$R_3 : s \in [30, 80] \rightarrow deny$$

su propuesta no será capaz de detectar el enmascaramiento de la regla R_3 debido a la unión de las reglas R_1 y R_2. En segundo lugar, los autores no cubren, intencionadamente, una reescritura automática de la política de filtrado, una vez descubiertas las anomalías en la configuración. De este modo, será el administrador encargado del dispositivo cortafuegos el responsable de realizar los cambios finales.

3. Nuestra propuesta

El objetivo principal de los siguientes algoritmos es el descubrimiento y eliminación de redundancia y enmascaramiento en políticas de control de acceso de dispositivos o sistemas cortafuegos.

Nuestra propuesta se basa en un proceso de auditoría sobre un conjunto de reglas de filtrado inicial, R. Dicho proceso alertará de las anomalías detectadas y proporcionará un conjunto de reglas de filtrado final, libre de anomalías. La información utilizada durante el análisis es el siguiente. En primer lugar, el conjunto inicial

de reglas de filtrado R será proporcionado como una lista encadenada de reglas, de longitud n, donde n es igual a $count(R)$, y donde cada elemento es una tabla asociativa, cuyos campos, utilizados como clave de acceso al valor del elemento correspondiente, son: $condition$, $decision$, $shadowing$ y $redundancy$.

Para simplificar los algoritmos, supondremos que es posible acceder a la lista encadenada a través del operador R_i, donde i es la posición relativa respecto al tamaño de la lista inicial ($count(R)$). Asumiremos también que es posible añadir nuevos elementos a la lista de manera similar a cualquier otra variable, mediante sentencias $elemento \leftarrow valor$, así como eliminación de elementos a través de la inclusión de un conjunto vacío ($elemento \leftarrow \emptyset$). El orden interno de la lista encadenada R mantendrá el orden relativo entre reglas – es decir, la prioridad de cada regla en el conjunto de reglas R.

A su vez, cada elemento $R_i[condition]$ es una tabla indexada de tamaño p, que contiene el conjunto de condiciones de cada regla. Cada elemento $R_i[decision]$ es una expresión booleana cuyo valor está dentro del rango $\{accept, deny\}$; cada elemento $R_i[shadowing]$ es una expresión booleana en $\{true, false\}$; cada elemento $R_i[redundancy]$ es otra expresión booleana en $\{true, false\}$. Estas dos últimas variables serán inicializadas, por defecto, a $false$.

Por razones de claridad, dividimos el proceso global de análisis en tres algoritmos distintos. El primer algoritmo es una función auxiliar, cuya entrada son dos reglas, A y B. Una vez ejecutada, esta función auxiliar retornará una nueva regla, C, cuyo conjunto de atributos de condición corresponde a la exclusión de los atributos de las condiciones de A sobre B. Para simplificar la representación de este primer algoritmo, utilizamos la notación A_i como una abreviación de la variable $A[condition][i]$; y la notación B_i como una abreviación de la variable $B[condition][i]$ – donde i está dentro del rango $[1, p]$.

El segundo algoritmo implementa una función booleana dentro de $\{true, false\}$. A su vez, esta función aplica internamente una llamada recursiva al algoritmo anterior (función *exclusion*) sobre un subconjunto de reglas de filtrado. El objetivo es comprobar si la regla recibida como parámetro puede ser eliminada del subconjunto de reglas, también recibido como parámetro, sin que la política de filtrado de tal subconjunto varíe.

El tercer algoritmo realiza el proceso completo de detección y eliminado de redundancia y enmascaramiento, y se compone de dos fases claramente diferenciadas. Durante la primera, se realiza un recorrido descendente sobre el conjunto de reglas R, realizando el proceso de reescritura – mediante una llamada recursiva a la función *exclusion* – sobre aquellas reglas que tengan una decisión diferente.

```
1  begin
2      C[condition] ← ∅;
3      C[decision] ← B[decision];
4      C[shadowing] ← false;
5      C[redundancy] ← false;
6      forall the elements in A[condition] and B[condition] do
7          if ((A_1 ∩ B_1) ≠ ∅ and (A_2 ∩ B_2) ≠ ∅ and ... and (A_p ∩ B_p) ≠ ∅) then
8              C[condition] ← C[condition] ∪
9              {(B_1 − A_1) ∧ B_2 ∧ ... ∧ B_p,
10             (A_1 ∩ B_1) ∧ (B_2 − A_2) ∧ ... ∧ B_p,
11             (A_1 ∩ B_1) ∧ (A_2 ∩ B_2) ∧ (B_3 − A_3) ∧ ... ∧ B_p,
12             ...
13             (A_1 ∩ B_1) ∧ ... ∧ (A_{p−1} ∩ B_{p−1}) ∧ (B_p − A_p)};
14         else
15             C[condition] ← (C[condition] ∪ B[condition]);
16     return C;
17 end
```

Algoritmo 1: `exclusion`(B,A)

```
1  begin
2      test ← false;
3      i ← 1;
4      temp ← r;
5      while ¬test and (i ≤ count(R)) do
6          temp ← exclusion(temp, R_i);
7          if temp[condition] = ∅ then
8              test ← true;
9          i ← (i + 1);
10     return test;
11 end
```

Algoritmo 2: `testRedundancy`(R,r)

```
1  n ← count(R);
2  /*Phase 1*/
3  for i ← 1 to (n − 1) do
4      for j ← (i + 1) to n do
5          if R_i[decision] ≠ R_j[decision] then
6              R_j ← exclusion (R_j,R_i);
7              if R_j[condition] = ∅ then
8              R_j[shadowing] ← true;
9  /*Phase 2*/
10 for i ← 1 to (n − 1) do
11     R_a ← {r_k ∈ R | n ≥ k > i and
12         r_k[decision] = r_i[decision]};
13     if testRedundancy (R_a,R_i) then
14         R_i[condition] ← ∅; R_i[redundancy] ← true;
15     else
16         for j ← (i + 1) to n do
17             if R_i[decision]=R_j[decision] then
18                 R_j ←exclusion (R_j,R_i);
19                 if (¬R_j[redundancy] and
20                 R_j[condition] = ∅) then
21                  R_j[shadowing] ← true;
```

Algoritmo 3: audit(R)

Obsérvese que esta etapa del algoritmo, donde se producirá una detección y eliminación de reglas enmascaradas, se realiza justo antes de la detección y eliminación de reglas redundantes. El conjunto de reglas resultante de esta primera etapa servirá de entrada a la segunda fase, de nuevo, en un recorrido descendente. Durante esta segunda fase, se realizará una llamada a la función *testRedundancy*, que permitirá diferenciar si la exclusión total de una regla es debida a una redundancia o a un enmascaramiento.

3.1. Corrección de los algoritmos

Lema 1 *Sea R el conjunto inicial de reglas de filtrado; sean $R_i : condition_i \rightarrow decision_i$ y $R_j : condition_j \rightarrow decision_j$ dos reglas de filtrado del conjunto R; sea R'_j el resultado de aplicar $exclusion(R_j, R_i)$. El conjunto $\{R_i, R_j\}$ es equivalente a $\{R_i, R'_j\}$.*

Demostración. Sea $R_i[condition] = A_1 \wedge A_2 \wedge ... \wedge A_p$ y $R_j[condition] = B_1 \wedge B_2 \wedge ... \wedge B_p$ las condiciones de dos reglas de filtrado R_i y R_j dentro del conjunto R.

Demostrar la equivalencia entre los pares de reglas $\{R_i, R_j\}$ y $\{R_i, R'_j\}$, en el caso que $(A_1 \cap B_1) = \emptyset$ o $(A_2 \cap B_2) = \emptyset$ o ... o $(A_p \cap B_p) = \emptyset$, es decir, $exclusion(R_j, R_i) = R_j$, es trivial.

Supongamos ahora que $(A_1 \cap B_1) \neq \emptyset$ y $(A_2 \cap B_2) \neq \emptyset$ y ... y $(A_p \cap B_p) \neq \emptyset$. Si aplicamos en este segundo caso ambas reglas, siendo R_i prioritaria en orden a R_j, podemos afirmar, por consiguiente, que la regla R_j será aplicada a un paquete determinado si dicho paquete satisface $R_j[condition]$ pero no $R_i[condition]$ – ya que, de lo contrario, la regla R_i se aplicará primero.

Por otro lado, obsérvese que $R_j[condition] - R_i[condition]$ es equivalente a:

$$
\begin{array}{l}
(B_1 - A_1) \wedge B_2 \wedge ... \wedge B_p \text{ o} \\
(A_1 \cap B_1) \wedge (B_2 - A_2) \wedge ... \wedge B_p \text{ o} \\
(A_1 \cap B_1) \wedge (A_2 \cap B_2) \wedge (B_3 - A_3) \wedge ... \wedge B_p \text{ o} \\
... \\
(A_1 \cap B_1) \wedge ... \wedge (A_{p-1} \cap B_{p-1}) \wedge (B_p - A_p)
\end{array}
$$

que corresponde a la condición de la regla $R'_j = exclusion(R_j, R_i)$. De este modo, si la regla R_j se aplica a un paquete determinado dentro de $\{R_i, R_j\}$, entonces la regla R'_j también será aplicada a dicho paquete dentro de $\{R_i, R'_j\}$.

En cambio, si la regla R'_j se aplica a un determinado paquete dentro de $\{R_i, R'_j\}$, significaría que tal paquete satisface $R_j[condition]$ pero no $R_i[condition]$, lo que nos lleva a afirmar que R_j también se aplica a este paquete dentro de $\{R_i, R_j\}$.

Puesto que en el algoritmo $exclusion$, $R'_j[decision]$ se convierte en $R_j[decision]$, podemos concluir que $\{R_i, R_j\}$ es equivalente a $\{R_i, R'_j\}$. □

Teorema 2 *Sea R un conjunto de reglas de filtrado y sea $Tr(R)$ el resultado de aplicar $audit(R)$. Los conjuntos R y $Tr(R)$ son equivalentes.*

Demostración. Sea $Tr'_1(R)$ el conjunto de reglas resultante de aplicar la primera fase de $audit(R)$. Puesto que $Tr'_1(R)$ se deriva de R aplicando recursivamente la transformación $exclusion(R_j, R_i)$ sobre algunas reglas R_j del conjunto R, es evidente concluir, tras la demostración anterior, que $Tr'_1(R)$ es equivalente a R.

Por otro lado, moviéndonos ahora a la segunda fase de $audit(R)$, y tomando una regla R_i tal que $testRedundancy(R_i)$ retorne $true$, podemos constatar que $R_i[condition]$ puede ser derivada a partir de un subconjunto de reglas S que cumplen las siguientes condiciones: (1) apuntan a la misma decisión que R_i; (2) son prioritarias en orden a R_i.

Puesto que cada regla R_j con una decisión distinta a la decisión del subconjunto de reglas S habrá sido previamente retirada durante la primera fase del algoritmo, podemos concluir que la regla R_i es definitivamente redundante y puede ser, por tanto, retirada de la política de filtrado sin que se produzca ningún cambio en la decisión final de ésta. Por ello, podemos concluir, nuevamente, que se preserva la equivalencia también en este caso. Por último, si el resultado de $testRedundancy(R_i)$ retorna $false$, la transformación será, por consiguiente, la aplicación de $exclusion(R_j, R_i)$ sobre algunas reglas R_j, preservando nuevamente la equivalencia en este tercer caso.

Por todo ello, podemos afirmar que $Tr'(R)$ es equivalente a $Tr'_1(R)$ que, a su vez, es equivalente a R. □

Lema 3 *Sean $R_i : condition_i \rightarrow decision_i$ y $R_j : condition_j \rightarrow decision_j$ dos reglas de filtrado del conjunto R; sea $R'_j \leftarrow exclusion(R_j, R_i)$. Las reglas R_i y R'_j no serán nunca aplicadas simultáneamente sobre un mismo paquete.*

Demostración. Obsérvese que la regla R'_j solo se aplica cuando la regla R_i no lo hace. Por consiguiente, si la regla R'_j se coloca por delante de la regla R_i, ésto no afectará a la decisión final de la política de filtrado, ya que la regla R'_j solo se aplicará sobre aquellos paquetes que no sean asociados a la regla R_i. □

Teorema 4 *Sea R un conjunto de reglas de filtrado y sea $Tr(R)$ el resultado de aplicar $audit(R)$. El orden de reglas de $Tr(R)$ no es relevante.*

Demostración. Para cualquier pareja R_i y R_j, donde R_i se aplica con anterioridad a R_j, R_j es reemplazada por R'_j al aplicar recursivamente R_j tras la llamada a $exclusion(R_j, R_k)$ para cualquier $k < j$. Por ello, y aplicando recursivamente el principio enunciado en el lema 3, es posible intercambiar el orden de las reglas R'_i y R'_j dentro de $Tr(R)$ sin provocar ningún cambio en la política de filtrado. □

Teorema 5 *Sea R un conjunto de reglas de filtrado y sea $Tr(R)$ el resultado de aplicar $audit(R)$. El conjunto de reglas $Tr(R)$ es libre tanto de redundancia como de enmascaramiento.*

Demostración. Puesto que dentro de $Tr(R)$, cada regla es completamente independiente del resto de reglas, si consideramos una regla R_i proveniente de $Tr(R)$ tal que $R_i[condition] \neq \emptyset$, podemos asegurar que esta arregla será aplicada sobre cualquier paquete que satisfaga $R_i[condition]$. Por tanto, esta regla no será nunca enmascarada por ninguna otra regla del conjunto.

De manera similar, R_i no es redundante ya que en caso de ser eliminada del conjunto de reglas de $Tr(R)$, la política de filtrado del conjunto será diferente, pues no existe ninguna otra regla en el conjunto $Tr(R)$ que se aplique al mismo tráfico que satisface $R_i[condition]$. □

4. Evaluación

En esta sección analizamos la complejidad teórica de nuestra propuesta y se presenta una breve evaluación del rendimiento ofrecido por la primera implementación realizada de nuestros algoritmos.

4.1. Complejidad de los algoritmos

En el peor de los casos, el algoritmo 3 presentado en este artículo puede llegar a generar un número de reglas derivadas muy elevado. Si el número de reglas inicial es de 2, por ejemplo, con p atributos de condición cada regla, la segunda regla puede llegar a ser reemplazada, en el peor de los casos, por p nuevas reglas, incrementando el conjunto inicial de reglas de 2 a $p + 1$.

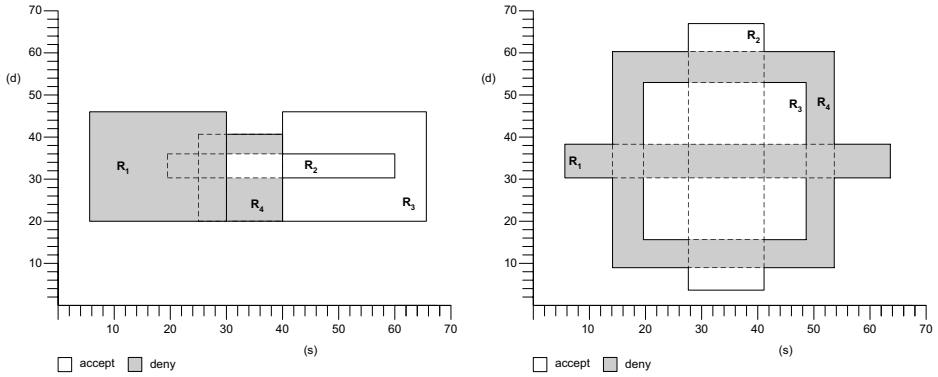
Asumiendo que el número de reglas sea n, donde $n > 2$, con p atributos cada regla, esto nos lleva a concluir que, en el primer paso de reescritura del algoritmo, cada regla, a excepción de la primera, puede llegar a ser reemplazada por p nuevas reglas.

En el segundo paso de reescritura del algoritmo, las p reglas que reemplazan a la segunda regla serán combinadas con las p nuevas reglas que reemplazan a cada regla, de la 3 a la n. Así pues, cada regla, de 3 a n, es susceptible de ser reemplazada en el peor de los casos por p^2 nuevas reglas. En el tercer paso de reescritura, las p^2 reglas correspondientes a la regla 3 son combinadas con las p^2 reglas correspondientes a las reglas 4 a n, incrementando éstas hacia p^3 nuevas reglas cada una. Y así de manera sucesiva.

Por lo tanto, en el peor de los casos, si el número inicial de reglas es igual a n (donde $n > 2$) con p atributos, al finalizar la ejecución del algoritmo 3 podemos llegar a obtener $1 + p + p^2 + \ldots + p^{n-1}$ reglas, es decir, $\frac{p^n-1}{p-1}$ reglas.

Por tanto, podemos constatar que la complejidad teórica del algoritmo 3 es realmente elevada. Sin embargo, en los experimentos realizados (ver siguiente apartado), los resultados han estado siempre extraordinariamente lejos de tal incremento en el número de reglas. Los motivos son los siguientes. En primer lugar, tan solo

los atributos *source* y *destination* suelen solaparse y ejercer una mala influencia en la complejidad del algoritmo. El resto de atributos, como el protocolo o los números de puerto de origen o destino, son, por lo general, iguales o completamente distintos cuando combinamos distintas reglas de filtrado.



(a) Ejemplo de un caso normal (b) Ejemplo del peor de los casos

Figura 1. Ejemplo de solapamiento de reglas

En segundo lugar, los administradores suelen incluir, de forma expresa, casos de solapamiento para expresar excepciones. Esta situación es más cercana al caso *normal*, representado en la figura 1(a), que al peor de los casos representado en la figura 1(b).

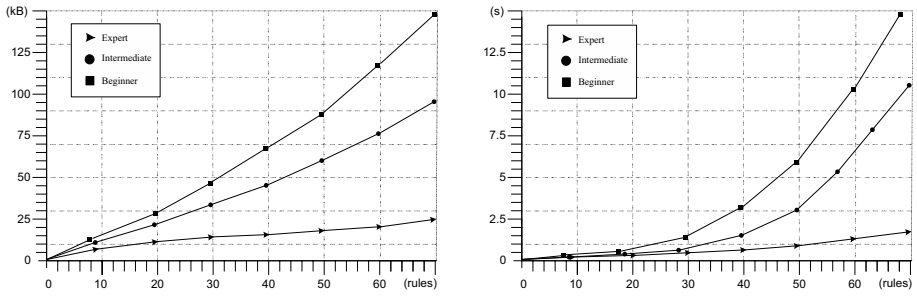
Por último, cuando situaciones de redundancia o enmascaramiento son detectados por el algoritmo, dichas reglas desaparecen, lo cual ayuda a reducir significativamente la complejidad del algoritmo.

4.2. Rendimiento

En esta sección presentamos una evaluación del rendimiento de MIRAGE (MIsconfiguRAtion manaGEr), un prototipo *software* que implementa una primera versión de nuestros algoritmos. MIRAGE ha sido desarrollado en lenguaje PHP, un lenguaje de propósito general, especialmente indicado para el desarrollo de servicios web, que puede ser combinado con el lenguaje HTML para la construcción de interfaces gráficas de usuario [4]. De este modo, MIRAGE puede ser utilizado

de forma local o remota, mediante la utilización de un servidor de HTTP y un navegador web.

Para evaluar el rendimiento y eficiencia de este primer prototipo, fueron simulados, en una primera fase de los experimentos, distintos conjuntos de reglas de filtrado para redes IPv4, de acuerdo a los siguientes tres perfiles de administrador: aprendiz, intermedio y experto – donde la probabilidad de solapamiento en cada caso incrementa de un 5 % a un 90 %. A continuación, en una segunda fase, este conjunto de reglas de filtrado fue utilizado como entrada de MIRAGE.



(a) Espacio requerido en memoria (b) Tiempo de cómputo

Figura 2. Evaluaciones de memoria y tiempo de cómputo

El total de estos experimentos fue llevado a cabo en un único equipo con procesador Intel-Pentium M 1.4 GHz, con 512 MB de memoria RAM, funcionando sobre un sistema operativo Debian GNU/Linux 2.6.8, con un servidor de HTTP Apache/1.3 y un intérprete PHP/4.3 configurado.

Durante estos experimentos, se realizaron medidas del espacio de memoria ocupado por el prototipo, así como el tiempo de cómputo requerido para la finalización de los algoritmos 1, 2 y 3. Los resultados de estas medidas son presentados en las figuras 2(a) y 2(b).

Aunque estos resultados reflejan unos requerimientos de memoria y tiempo de proceso un tanto elevados, consideramos que son del todo razonables para la realización de un análisis *off-line*, puesto que tal análisis no será una parte crítica del rendimiento del dispositivo analizado.

5. Conclusiones

Hay dos maneras de tratar la existencia de errores o anomalías en la política de filtrado de un sistema cortafuegos. Un primer enfoque se basa en la utilización de un modelo formal, como el modelo presentado en [5], con el cual expresar de manera unívoca la política de control de acceso a la red, y generar, a partir de ahí, la configuración específica para cada uno de los dispositivos de la red.

La ventaja principal de este primer enfoque es la gran confianza depositada en la utilización de un modelo formal, y su traducción en forma de configuración de un dispositivo cortafuegos específico. Sin embargo, este enfoque formal no puede descartar que futuras anomalías sean introducidas directamente en el sistema cortafuegos, por parte del propio administrador.

Un segundo enfoque – como el que se ha presentado en este artículo – se basa en la aplicación directa de un proceso de análisis contra la configuración del sistema a analizar. Dicho proceso tratará de detectar los errores de configuración existentes y recomendar al administrador su modificación o eliminación.

En nuestro caso, el proceso de auditoría se basa en la existencia de relaciones entre los atributos de condición de las reglas de filtrado. Nuestra propuesta utiliza un proceso de transformación de reglas, que hará derivar una política de filtrado inicial, potencialmente mal configurada, hacia una política de filtrado equivalente, totalmente libre de anomalías.

Algunas de las ventajas de nuestra propuesta son las siguientes. En primer lugar, nuestro proceso de transformación verifica que las reglas resultantes son totalmente independientes entre sí. De lo contrario, cualquier regla considerada inútil durante el proceso de análisis, será notificada al administrador, y apartada del conjunto de reglas resultante.

En segundo lugar, la posibilidad de proveer estas notificaciones al responsable de administración, permitirá que sea el operador encargado de la administración del dispositivo el último responsable en verificar la consistencia del proceso, y verificar si finalmente las reglas apartadas deben desaparecer o no de la configuración.

La completa independencia entre reglas, además, permite al administrador realizar una segunda transformación del conjunto de reglas, pudiendo establecer una política de filtrado abierta (donde las reglas solo indican prohibiciones) o cerrada (donde las reglas solo indican permisos). Una vez realizada esta segunda transformación, el administrador tendrá una visión más nítida del tráfico aceptado o rechazado por cada dispositivo cortafuegos.

Respecto al posible aumento en el número inicial de reglas de filtrado, debido a nuestro proceso de transformación, se ha comprobado de forma empírica como dicho aumento no llega a producirse en la magnitud esperada – tal y como se ha expuesto en el apartado 4.2. Además, es importante recordar que el incremento en el número de reglas de un sistema cortafuegos no se traduce sistemáticamente en una degradación de su rendimiento. El uso de algoritmos de clasificación independientes del número de reglas es una buena solución tanto para el problema que motiva nuestra propuesta, como para el despliegue de sistemas cortafuegos modernos. El algoritmo de clasificación propuesto en [15], por ejemplo, nos ofrece la posibilidad de procesar el tráfico de red sobre una política de filtrado específica, sin depender del número de reglas – pues depende tan solo del número y tamaño de los atributos de condición.

La puesta en funcionamiento de nuestro algoritmos en un prototipo software, brevemente analizado en el apartado 4.2, demuestra la viabilidad de nuestro trabajo. Aunque los resultados experimentales muestran que nuestros algoritmos tienen fuertes restricciones de memoria y tiempo de cómputo, creemos que estos requisitos son razonables para la realización de análisis *off-line*, ya que el funcionamiento crítico del sistema cortafuegos no dependerá de dicho análisis.

Como trabajo futuro, estamos considerando extender nuestra propuesta para el análisis de políticas de filtrado más complejas. El trabajo iniciado en [6] – y su continuación en el presente artículo – se basa en la hipótesis que tan solo un dispositivo cortafuegos realiza el control de acceso de la red. Nuestra motivación es extender esta propuesta hacia configuraciones con más de un dispositivo de control, es decir, con un control de acceso distribuido. El objetivo es realizar un proceso de correlación de información, tratando de ver como interactúan las distintas políticas de seguridad de cada dispositivo, desde un punto de vista global. Aunque existen actualmente propuestas para este propósito como, por ejemplo, la presentada en [2], consideramos los resultados de dicha propuesta como incompletos y, en según que situaciones, incorrectos.

En paralelo, estamos estudiando también extender el proceso de análisis de nuestra propuesta para la búsqueda y eliminación de anomalías entre distintos dispositivos de seguridad, es decir, no únicamente entre dispositivos cortafuegos, sino también configuraciones donde aparezcan sistemas de detección de intrusos o sistemas para la construcción de redes privadas virtuales. Consideramos que hay una gran similitud entre el análisis proporcionado en este artículo, y el análisis necesario para solucionar este nuevo enfoque.

Agradecimientos

El presente trabajo ha sido parcialmente financiado mediante el proyecto *ACI DESIRS* del Ministerio Francés de la Investigación, el proyecto TIC2003-02041 del Ministerio Español de Ciencia y Tecnología, y las becas *2003FI126* y *2005BE77* del departamento DURSI de la Generalitat de Catalunya.

Referencias

1. Adiseshu, H., Suri, S., and Parulkar, G. (2000). Detecting and Resolving Packet Filter Conflicts. *In 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, pages 1203–1212.
2. Al-Shaer, E. S., Hamed, H. H., and Masum, H. (2005). Conflict Classification and Analysis of Distributed Firewall Policies In *IEEE Journal on Selected Areas in Communications*, 23(10).
3. Bartal, Y., Mayer, A. J., Nissim, K., and Wool, A. (1999). Firmato: A novel firewall management toolkit. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 17–31.
4. Castagnetto, J., Rawat, H., Schumann, S., Scollo, C., and Veliath, D. (1999). *Professional PHP Programming*. Wrox Press Inc, ISBN 1-86100-296-3, 909 pages.
5. Cuppens, F., Cuppens-Boulahia, N., Sans, T. and Miege, A. (2004). A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust*, 203–218.
6. Cuppens, F., Cuppens-Boulahia, N., and García-Alfaro, J. (2005). Detection and Removal of Firewall Misconfiguration. In *Proceedings of the 2005 IASTED International Conference on Communication, Network and Information Security*, 154–162.
7. Eppstein, D. and Muthukrishnan, S. (2001). Internet packet filter management and rectangle geometry. In *Proceedings of the 12th annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 827–835.
8. Frantzen, M., Kerschbaum, F., Schultz, E., and Fahmy, S. (2001). A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Computers and Security*, 20(3):263–270.
9. Gupta, P. (2000). *Algorithms for Routing Lookups and Packet Classification*. PhD Thesis, Department of Computer Science, Stanford University.
10. Guttman, J. D. (1997). Filtering postures: Local enforcement for global policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–129.
11. Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., and Frantzen, M. (2003). Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214–232.
12. Liu, A. X. and Gouda, M. G. (2004). Diverse firewall design. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 595–604.
13. Liu, A. X. and Gouda, M. G. (2005). Complete Redundancy Detection in Firewalls. In *Proceedings of 19th Annual IFIP Conference on Data and Applications Security*, pages 196–209.
14. Mayer, A. J., Wool, A., and Ziskind, E. (2000). Fang: A firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 177–187.
15. Paul, O., Laurent, M., and Gombault, S. (2000). A full bandwidth ATM Firewall. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, pages 206–221.
16. Srinivasan, V., Suri, S., and Varghese, G. (1999). Packet classification using tuple space search. *Computer ACM SIGCOMM Communication Review*, pages 135–146.