Modelo de arreglo de árboles para detección de cambios no autorizados

A tree array model for detection of unauthorized changes

Manuel Emilio Zúñiga Rodríguez[1] Br. & Marta Eunice Calderón Campos[2] MSc.
1. Universidad de Costa Rica, San Pedro, Costa Rica
2. Universidad de Costa Rica, Docente, Escuela de Ciencias de la Computación e Informática, San Pedro, Costa Rica
mezr@costarricense.cr; marta.calderon@ecci.ucr.ac.cr

Recibido para revisión 30 de Septiembre de 2008, aceptado 25 de Agosto de 2009, versión final 08 de Septiembre de 2009

Resumen— En este artículo describimos un mecanismo de detección de cambios en un conjunto de datos a proteger. El mecanismo consiste en generar un conjunto de detectores a partir de cadenas de caracteres propias del conjunto de datos. La estructura de datos usada para el almacenamiento de detectores es un arreglo de árboles. Cada árbol del arreglo representa los detectores para una secuencia de r caracteres continuos dentro de una hilera de tamaño l del conjunto de datos. Cada nodo en un árbol representa un carácter en un detector. Para detectar un cambio, recorremos los nodos en la estructura de árbol para una hilera de caracteres dada y comparamos ésta con los detectores. Detectamos un cambio cuando no encontramos una ruta coincidente desde la raíz de un árbol hasta una hoja. Pusimos a prueba la eficacia del mecanismo propuesto con experimentos que consistieron en la fase de generación de todas las cadenas posibles para un alfabeto binario y la fase de detección, o sea, la clasificación de un conjunto de hileras de caracteres como hileras propias o no propias. Obtuvimos los mejores resultados en eficacia cuando *r* es igual a l.

Palabras Clave— Seguridad, Detección de Cambios, Generación de Detectores, Sistemas Inmunes Artificiales.

Abstract— This paper describes a mechanism for detection of changes in a data set to be protected or monitored. The mechanism is based on the creation of detectors derived from self character strings from the data set. The data structure used for storing detectors is a tree array. Each tree in the array stores detectors for a sequence of r adjacent characters within a l-character string extracted from the data set. Each node in a tree represents a character in a detector. In the monitoring phase, changes in the data set are detected going across nodes of the tree array. A character string being analyzed is compared with detectors in the tree array. A change is detected when no matching route from the root to its leaves is found. Effectiveness in detector generation and change detection of the proposed mechanism is tested with experiments consisting of generating all posible character chains for a binary alphabet and executing a detection or monitoring phase, that is, classifying a set of character strings as proper or non-proper. The best non-proper string detection results were obtained when *r* is equal to l.

Keywords— Security, Change Detection, Detector Generation, Artificial Immune Systems.

I. INTRODUCCIÓN

Es necesario renovar y revisar los mecanismos de seguridad aplicados a un sistema de software o conjunto de datos constantemente, dado que cada día es posible encontrar nuevas vulnerabilidades en un sistema dado, ya sea por nuevos componentes agregados (nuevo código, cambio en estructura) o por defectos no hallados con anterioridad en los componentes existentes. Los defectos no hallados pueden ser fuente de ataques imprevistos, y por lo tanto debilidades en la seguridad del sistema. Algunos mecanismos de seguridad, como la detección de intrusiones, buscan proteger un sistema detectando uso no autorizado, mal uso o abuso de un sistema, ya sea mediante anomalías en perfiles de uso o en la detección previa al abuso del sistema [1]. Los detectores de anomalías requieren recolección de datos que ayuden a identificar el comportamiento malicioso de un usuario, lo cual no es siempre posible. Por otro lado, los sistemas de detección de abuso requieren la intervención de un experto en seguridad, encargado de definir mediante reglas los ataques conocidos por el sistema [7].

Los detectores de abuso no son capaces de reconocer nuevos tipos de ataques. Por eso se ha recurrido a otros mecanismos, como la imitación del sistema inmunológico humano, el cual es capaz de proteger el cuerpo de sustancias extrañas, sin previo conocimiento de las mismas, mediante diversos mecanismos [2, 4, 6]. En el cuerpo humano, las células T son las encargadas de la detección de sustancias extrañas. Estas células siguen un proceso de selección negativa en su maduración, antes de iniciar su trabajo en la detección de sustancias extrañas [4, 6]. Básicamente, la selección negativa evita que las células T autoinmunes, o sea, identificadoras de lo propio como extraño, lleguen a la madurez y se incorporen al sistema inmunológico. Se han planteado varios mecanismos de detección de cambios no autorizados en sistemas computacionales basados en la manera en que funciona el sistema inmunológico humano, o sea, basados en el mecanismo de selección negativa del sistema inmunológico [3, 5, 6]. En estos casos se habla de cadenas -de caracteres- propias (particiones de los datos a proteger) y de cadenas no propias. Los mecanismos de detección generan un conjunto de detectores que sirven para determinar si una cadena es propia o no. Si algún detector hace pareja con la cadena examinada, se determina que la cadena es no propia. El proceso de generación de detectores en estos enfoques puede ser costoso en términos de tiempo.

No se han propuesto mecanismos de detección que utilicen las cadenas propias para detectar si una cadena cualquiera es reconocida como propia o no propia. El objetivo de este artículo es presentar un mecanismo de detección de cambios en el que el primer paso consiste en definir las cadenas propias de los datos a proteger y, a partir de ellas, el segundo paso es determinar si una cadena cualquiera es propia o no propia.

La estructura de este artículo se describe a continuación. En la Sección 2 se describen trabajos anteriores relacionados con la definición de mecanismos de seguridad basados en el mecanismo de selección negativa del sistema inmunológico humano. En la Sección 3 se detalla el mecanismo de detección de cambios propuesto. En la Sección 4 se muestran los experimentos realizados para medir la eficacia del mecanismo propuesto y en la Sección 5 los resultados obtenidos. La Sección 6 describe el trabajo que queda pendiente. Finalmente, en la Sección 7 se presentan las conclusiones de esta investigación.

II. TRABAJOS RELACIONADOS

En [6] se presenta un algoritmo de detección de cambio en un sistema basado en el mecanismo de selección negativa del sistema inmunológico humano. El algoritmo consiste en generar de manera aleatoria un conjunto de detectores para una colección de cadenas propias. Una cadena propia es una de las particiones de los datos a proteger. Cada cadena tiene una longitu de l carácteres. Los detectores son los encargados de identificar si una cadena o hilera de datos cualquiera es parte de la colección de cadenas propias. Un detector por selección negativa identifica una cadena como no propia si una secuencia de r carácteres contiguos en el detector hace pareja con la cadena que fue presentada al detector. La cantidad de carácteres necesarios para que un detector identifique una cadena de carácteres cualquiera se define como r, con $r < l$.

Dado el tiempo exponencial necesario para generar los detectores que requiere el mecanismo presentado en [6], en [5] se proponen dos algoritmos que requieren tiempo lineal de ejecución. En lugar de generar los detectores de manera aleatoria, estos dos algoritmos determinan la cantidad de detectores existentes para las hileras que empiezan con un patrón de r carácteres. Luego de determinar los posibles detectores, se seleccionan aquéllos válidos (que no reconozcan las hileras propias). Además, a diferencia del generador aleatorio de detectores, los algoritmos de tiempo lineal evitan la creación de detectores redundantes [3, 5]. Por otro lado, para estos algoritmos es necesario considerar también el tiempo requerido para generar los patrones válidos [3].

En [3] se presenta una variación del algoritmo de generación de detectores de manera aleatoria presentado en [6]. El algoritmo consiste en agregar mutación a los detectores aleatorios. La mutación se lleva a cabo en las partes del detector que hacen pareja con el conjunto de hileras propias, para alejar al detector de ese conjunto. En este caso, se introduce un parámetro que indica el número de veces que se mutará un detector candidato antes de descartarlo del conjunto de detectores. A pesar de que se continúa generando los detectores de manera aleatoria, la mutación introducida busca reducir la cantidad de detectores desechados durante el proceso de generación, y por ende disminuir el tiempo de generación de detectores. Se puede dar el caso de que la mutación aleje o acerque el detector al conjunto de hileras propias, lo que da un resultado similar a una generación aleatoria de detectores [3]. Información sobre las complejidades espaciales y temporales de varios algoritmos generadores de detectores se pueden encontrar en [3].

III. MECANISMO DE DETECCIÓN DE CAMBIOS PROPUESTO

A diferencia de los mecanismos generadores de detectores mostrados en [3, 5, 6], en este artículo presentamos un mecanismo que utiliza el conjunto de cadenas propias como base para la generación de los detectores. En lugar de generar detectores que no hagan pareja con las cadenas propias para encontrar las cadenas no propias, el mecanismo propuesto utiliza las cadenas propias para detectar si una cadena cualquiera es propia o no propia. El mismo mecanismo utilizado para almacenar los detectores sirve también para realizar el proceso de detección.

La idea del mecanismo de detección de cambios propuesto es generar un conjunto de detectores para un conjunto de datos a

proteger o monitorear. La estructura de datos propuesta para la generación y el almacenamiento de detectores es un arreglo de árboles. Cada árbol del arreglo representa los detectores para una secuencia de r carácteres continuos dentro de una hilera de tamaño l.

Una vez que un conjunto de detectores está generado y almacenado en el arreglo de árboles, sigue la fase de monitoreo o detección, la cual consiste en aplicar los detectores a un conjunto de hileras extraídas del conjunto de datos a proteger para determinar si ha ocurrido un cambio no autorizado, o sea, para determinar si las hileras son propias o no propias.

3.1. Estructura de datos utilizadas en el modelo

El mecanismo de detección de cambios propuesto en este artículo está basado en el algoritmo propuesto en [6]. Sin embargo, utiliza una estructura de arreglo de árboles A para la representación de los detectores. El tamaño del arreglo está dado por (l - r + 1) árboles. Este tamaño indica la cantidad de hileras de tamaño r que es posible formar a partir de una hilera s de tamaño l. Por ejemplo, dada la hilera 01001110, para r = 4 existen cinco hileras de tamaño r, a saber:

0100
1001
0011
0111
1110

Cada una de estas hileras es una ventana de r carácteres en la cadena original s. Utilizamos un árbol en el arreglo de árboles para representar cada ventana. El índice i del árbol dentro del arreglo está determinado por la posición del primer elemento de cada hilera h de r caracteres en la hilera original s. Empezamos a contar las posiciones o índices a partir del carácter que está más a la izquierda en la cadena s, iniciando en cero. La hilera h_i corresponde a la hilera de r bits que empieza en la posición i de la hilera s.

Un árbol a en el arreglo de árboles A está conformado por nodos que representan carácteres de una cadena de r bits. El árbol a_i posee en sus nodos los carácteres que conforman las cadenas h_i. Los nodos en cada árbol pueden pertenecer a distintas cadenas. Cada nodo posee una bandera booleana que indica si la ruta desde la raíz del árbol hasta ese nodo hace pareja con una hilera propia. En la Figura 1 se puede observar un árbol común para las hileras 010 y 011. Los nodos sombreados indican el final de un detector de una hilera propia. Además, cada árbol posee dos apuntadores: un apuntador al nodo raíz (R) y un apuntador al último nodo que ha sido agregado al árbol.

La cantidad de niveles o profundidad que posee un árbol en A está dado por r. Es decir, si recorremos el árbol desde la raíz hasta una de sus hojas (a r niveles de profundidad), podemos construir una cadena, que estaría formada por la concatenación

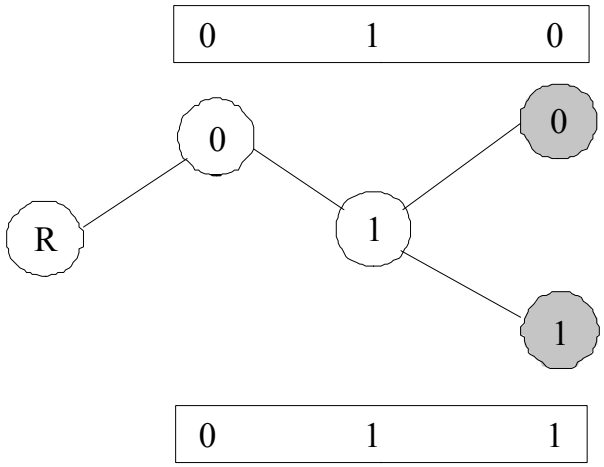

Figura 1. Representación de hileras de un árbol de profundidad r = 3

de los valores encontrados en los nodos recorridos. Un nodo n a profundidad j en un árbol representaría el caracter j ($0 <= j < r$) de una cadena de r bits. La raíz del árbol no cuenta como un nivel.

Nótese que el carácter de la posición 3 de la cadena s = 01001110 para r = 4 inicia la cadena h_3 = 0111. Sin embargo, ese mismo carácter también pertenece a las hileras h_0, h_1, y h_2. Esto quiere decir que en cada uno de los árboles a_0, a_1, a_2 y a_3 existe un nodo que representa el caracter de la posición 3 en s, en las posiciones *3, 2, 1* y *0*, respectivamente. En la Figura 2 se muestra el arreglo de árboles para la hilera 01001110 cuando l = 8 y r = 4.

Los árboles presentados en la Figura 2 representan en sus nodos únicamente una hilera, pues no poseen nodos que sean parte de varias hileras h. Sin embargo, tal como se observa en la Figura 1, cada uno de los árboles a puede representar varias hileras a la vez. El máximo de hileras representable por un árbol de profundidad r depende del tamaño del alfabeto de carácteres utilizado. Dado un alfabeto de tamaño m (el alfabeto posee m caracteres diferentes), la cantidad de hileras representable por un árbol de profundidad r se define como m^r. El alfabeto de caracteres indica cuáles elementos representan las entradas i en una hilera s. En las Figuras 1 y 2 se muestran hileras cuyo alfabeto es binario.

3.2. Generación del conjunto de detectores

Para generar un conjunto de detectores, dividimos los datos a proteger en cadenas o hileras de l carácteres. Cada una de estas hileras de tamaño l es una hilera propia. Es posible generar un conjunto de detectores tomando en cuenta todas las hileras propias o bien solamente un subconjunto de éstas. En este último caso, un arreglo de árboles tendrá menos nodos. Es posible realizar aleatoriamente la selección del subconjunto de las cadenas propias. Un parámetro de densidad d, donde $0 < d <= 1$, determina el tamaño del subconjunto. Utilizamos las hileras propias (todas o el subconjunto, según sea el caso) para agregar detectores de tamaño r al arreglo de árboles A, siguiendo

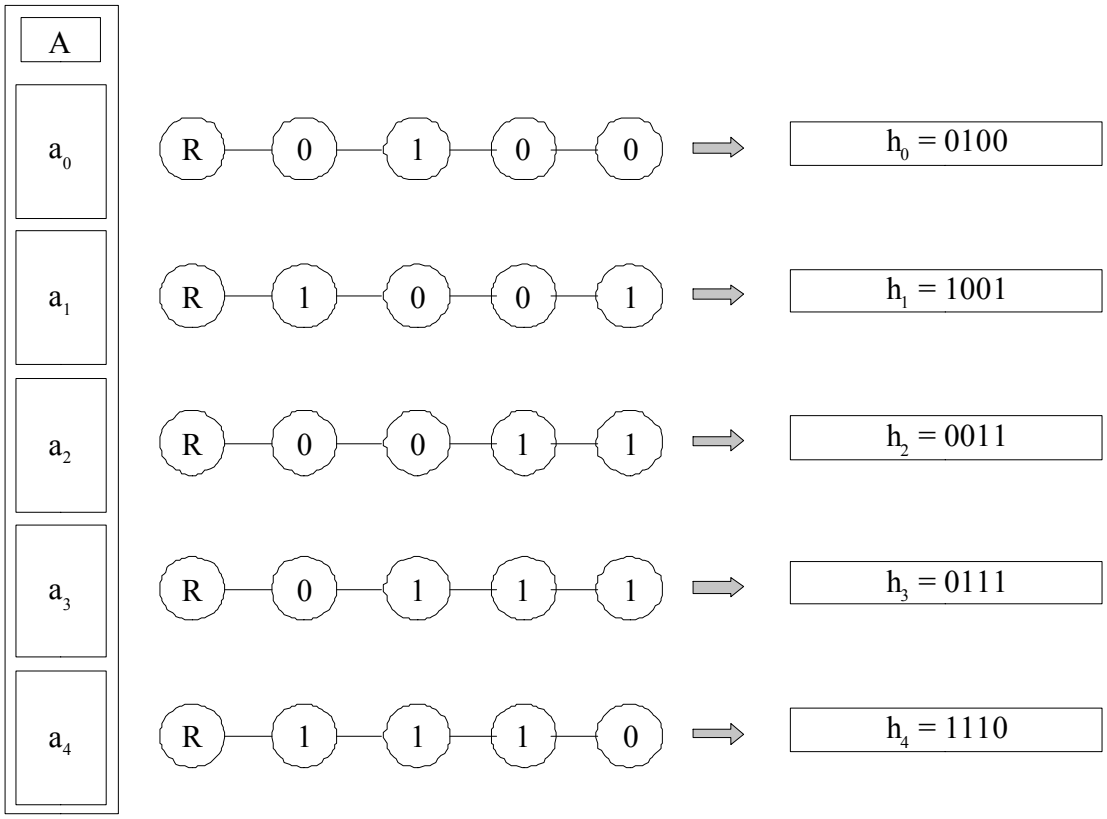

Figura 2. Estructura de arreglo de árboles para la cadena 01001110

los pasos expuestos a continuación para cada carácter c_i de la hilera propia s, con $0 <= i < l$:

1. Determinar en cuáles árboles ak de A debe haber un nodo para representar el carácter ci. Los valores de k están determinados por $(i - r + 1) <= k <= i$. Además, independendientemente de i, los valores válidos de k están limitados por $0 <= k < (l - r + 1)$. Es decir, el valor de k debe corresponder con una de las entradas en A. Por ejemplo para la iteración dada por $i = 1$, $r = 4$ y $l = 8$, los valores válidos de k en esa iteración son 0 y 1. Es decir, el carácter c_1 pertenece solamente a los árboles a_0 y a_1.

2. Modificar, para cada árbol a_k, el apuntador al último nodo agregado para que apunte al nodo raíz

3. Agregar el carácter c_i a los árboles a_k. Para ello, insertamos los nuevos nodos en los árboles de A, justo después del último nodo agregado. Además, actualizamos el apuntador al último nodo. Si ya existe un nodo en la ruta dada por el valor que estamos insertando, simplemente actualizamos el apuntador al último nodo agregado con la dirección del nodo existente.

En la Figura 3 se muestra la secuencia de pasos ejecutados para insertar dos hileras con $l = 4$ y $r = 2$. El nodo con borde grueso indica el nodo agregado. Los nodos sombreados indican el final de una hilera de r carácteres.

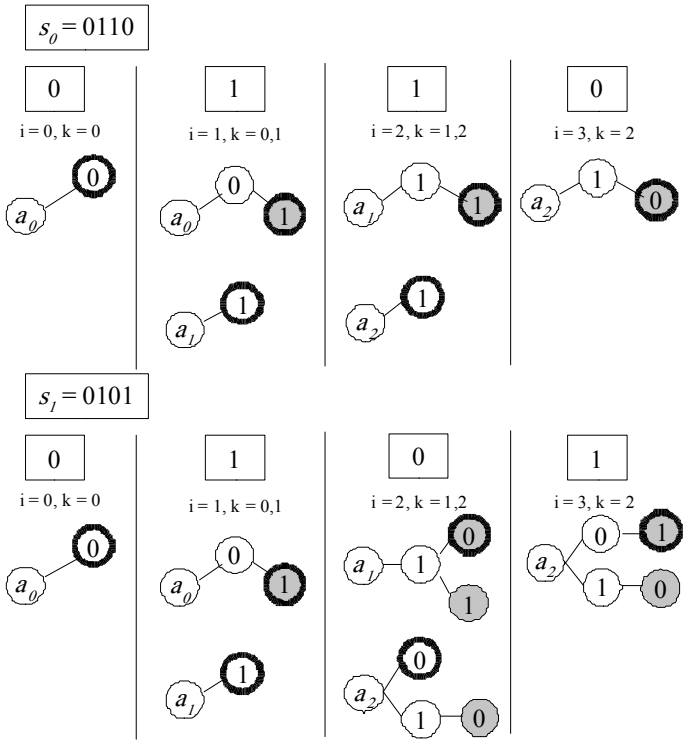

Figura 3. Secuencia de pasos para insertar hileras propias

Cuando cambia el conjunto de datos a proteger, es necesario volver a generar los detectores. Sin embargo, esta característica también es válida para mecanismos de detección basados en el mecanismo de selección negativa del sistema inmunológico humano.

3.3. Detección de cambios utilizando el modelo de arreglo de árboles

Una vez que se han insertado las hileras propias en el arreglo A, la estructura está lista para iniciar la fase de detección de hileras no propias. La comparación entre hileras no la hacemos de manera exacta. En otras palabras, para determinar la igualdad de dos cadenas, no realizamos una comparación entre todos sus carácteres, sino que verificamos que las cadenas posean *r* caracteres iguales. Si dos cadenas cualesquiera presentan *r* carácteres consecutivos iguales, podemos decir que ambas cadenas hacen pareja. La idea de la detección en el modelo de árbol consiste en determinar si una cadena dada es identificada como propia (hace pareja con otra cadena presente el árbol) o no propia (no existe una cadena que haga pareja).

En el momento de identificar una cadena, ejecutamos una secuencia de pasos similar a la necesaria para introducir una cadena propia, con la salvedad de que no agregamos nuevos nodos. Durante la detección, utilizamos la bandera booleana de los nodos para determinar el tipo de cadena que estamos analizando. Los pasos a seguir son:

1. Si la bandera está apagada y el nodo tiene hijos (aún no se hemos recorrido r carácteres), la cadena no está aún identificada (DESCONOCIDO).

2. Si la bandera está encendida, sabemos que para la secuencia de r carácteres analizados en ese árbol, la cadena es propia (PROPIA). Sin embargo, no sabemos si otras secuencias de r carácteres de la hilera presentan características no propias, por lo que no detenemos el proceso de detección. En el caso de que no haya más árboles por analizar, identificamos la cadena como propia.

3. Si la bandera está apagada y el nodo es una hoja, identificamos la cadena analizada como no propia (NO_PROPIA) y detenemos el proceso de detección, sin importar si hay más árboles por analizar. En realidad, llegamos a este tipo de nodos cuando seguimos una ruta inexistente en el árbol para indicar que una cadena no propia ha sido encontrada.

En la Figura 4 se presenta la secuencia de pasos para analizar la cadena 0111 con los árboles obtenidos del ejemplo de la Figura 3. El nodo con borde grueso indica el nodo analizado en la iteración. En la parte inferior de la Figura 4 se encuentra el análisis obtenido del nodo en los árboles. De ella se deriva que la hilera 0111 no es propia.

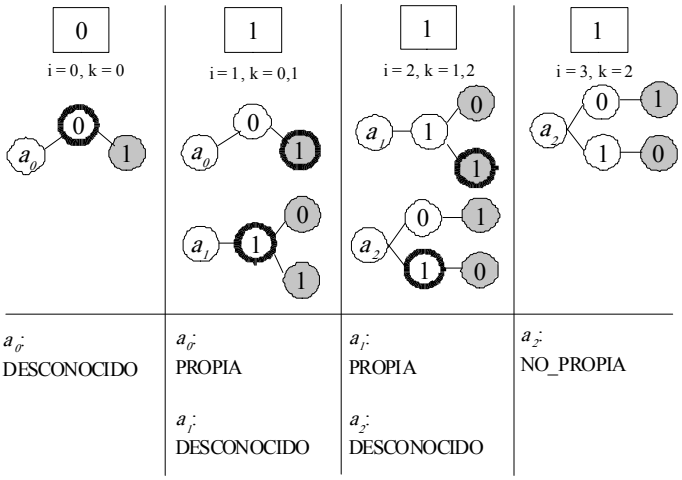

Figura 4. Secuencia de pasos ejecutada durante la identificación de una cadena

3.4. Complejidad temporal del modelo

A diferencia de los generadores de detectores presentados en [3, 5, 6], para generar el conjunto de detectores utilizando un módelo de árbol únicamente es necesario presentar las cadenas propias a la estructura.

Debido a que modificamos varios árboles a la vez por cada cadena presentada, el tiempo de inserción no depende únicamente de la cantidad de carácteres *l*, sino también de r, que es el parámetro que define la longitud de una instancia de hilera. Dado que tenemos $(l - r + 1)$ árboles en el arreglo y la cantidad de nodos agregados por árbol corresponde a *r*, el tiempo de inserción para una cadena de tamaño *l* está determinado, en el peor de los casos, por la función $r(l - r + 1)$, que es equivalente a $-r2 + r(l + 1)$.

Al ser una función cuadrática, los ceros indican el tiempo mínimo necesario para introducir una hilera en la estructura de árbol. Los ceros de la función son 0 y $(l + 1)$. Sin embargo, los ceros de la función no son valores válidos para r, que se encuentra en el rango $1 < r <= l$, con *r* número natural. Descartando valores inválidos de *r*, obtenemos los tiempos mínimos cuando $r = 1$ ó $r = l$, tal como se muestra en las fórmulas 1 y 2:

$$r = 1 \rightarrow r(l - r + 1) = 1(l - 1 + 1) = l \rightarrow O(l) \quad (1)$$

$$r = l \rightarrow r(l - r + 1) = l(l - l + 1) = l(1) = l \rightarrow O(l) \quad (2)$$

Por otro lado, el tiempo máximo se puede obtener de los ceros de la primera derivada de la función, como se muestra en las fórmulas 3 y 4 :

$$f' = 0 = -2r + (l + 1) \rightarrow -2r = -(l + 1) \rightarrow r = (l + 1) / 2 \quad (3)$$

$$r = (l + 1) / 2 \rightarrow 2r = (l + 1) \rightarrow r(l - r + 1) = -r^2 + r(l + 1) = -r^2 + r(2r) = 2r^2 - r^2 = r^2 \rightarrow O(r^2) \quad (4)$$

El máximo de la función podría no ser un valor entero y, por lo tanto, sería inválido. Por ejemplo, para l= 10, el mayor tiempo de inserción estaría dado por r = 5,5. Si obtenemos un valor no válido para r, podemos descartar los valores no naturales, utilizando el resultado de la función r= (l+ 1) / 2 para seleccionar los enteros inmediatamente inferior e inmediatamente superior al resultado. En el caso de l = 10, los máximos tiempos de inserción estarían dados por los valores válidos r= 5 y r= 6. Al seleccionar valores válidos para r, el resultado de aplicar la fórmula (3) para calcular la complejidad temporal podría no ser correcto, por lo que se utiliza la función original $r(l - r + 1)$. Un ejemplo de esta situación se muestra en la Tabla 1.

Tabla 1. Ejemplo para l=10 y varios valores de r en que r^2 no es el valor correcto para el tiempo máximo de inserción

r	r^2	$r(l - r + 1)$
5	25	30
6	36	30

A la hora de presentar una hilera para detectar un cambio no autorizado, seguimos un algoritmo similar al ejecutado para insertar una hilera propia. De ahí que el análisis de complejidad de tiempo es también aplicable a la fase de detección. El tiempo necesario para comparar todas las hileras propias contra todos los detectores está dado por la expresión S * D * $r(l - r + 1)$, donde S representa la cantidad de hileras propias y D el número de detectores. Debido a que la estructura de arreglos de árboles requiere únicamente la presentación de las hileras propias, y el costo asociado a una detección está dado por $r(l - r + 1)$, el costo de presentar todo el conjunto de hileras propias es S * $r(l - r + 1)$.

3.5. Complejidad espacial del modelo

Cada uno de los árboles a_i para las hileras h_i de r carácteres es capaz de representar mr cadenas distintas, donde m representa la cantidad de carácteres distintos en el alfabeto utilizado. Definimos la densidad d_i de la cadena h_i como la proporción de la cantidad de cadenas actualmente representadas en el árbol con respecto al total representable m^r, donde $0 < d_i <= 1$.

Tenemos ($l - r + 1$) árboles, y requerimos ($m^r - 1$) nodos internos (además de las hojas) para representar m^r cadenas. Por lo tanto, el espacio consumido en nodos está dado aproximadamente por la sumatoria de la fórmula 5:

$$\sum_{i=0}^{l-r+1} d_i(m^r + m^r - 1) \qquad (5)$$

IV. EXPERIMENTOS REALIZADOS

Los experimentos realizados para analizar la eficacia de la generación de detectores consistieron en generar todas las cadenas propias de un largo fijo l posibles para un alfabeto binario. Del total de cadenas propias generadas seleccionamos un subconjunto para representar la colección de cadenas propias. La selección de las cadenas que pertenecen a la colección se hizo de manera aleatoria. Determinamos el tamaño de la colección de cadenas propias utilizando un parámetro de densidad d.

Una vez seleccionado el conjunto de hileras propias, presentamos cada una a un arreglo de árboles para generar los detectores. Con el arreglo de árboles ya creado, procedimos a la fase de monitoreo o detección, que consistió en identificar como hileras propias o no propias un conjunto de hileras de caracteres extraídas de dos conjuntos: 1) un subconjunto de las hileras seleccionadas para la selección de los detectores y 2) un conjunto de hileras no seleccionadas para este proceso.

Los parámetros utilizados en las pruebas son el tamaño de las cadenas propias l, la cantidad de caracteres utilizados para determinar la igualdad r, la cantidad de elementos del alfabeto m y la densidad d del conjunto de hileras propias con respecto al total de hileras representables por my l. Las combinaciones de parámetros utilizadas en las pruebas se ejecutaron por 100 iteraciones. Los resultados que presentamos corresponden al promedio de la ejecución en las 100 iteraciones.

V. ANÁLISIS DE RESULTADOS

En la Tabla 2 se muestran los resultados obtenidos para un conjunto de hileras de tamaño l=10, con r = 8. Los parámetros de entrada utilizados son: el tamaño de las cadenas propias l, la cantidad de carácteres utilizados para determinar la igualdad r y la densidad d del conjunto de hileras propias con respecto al total de hileras representables por l. Los resultados obtenidos están representados por las columnas de la Tabla 2: el número de hileras propias P, el número de hileras propias detectadas

Tabla 2. Resultados obtenidos de la generación de detectores e identificación de hileras para los parámetros dados.

l	r	d	P	PD	NP	NPD	F	TIN	TID
10	8	0,1	102	102	922	739	20%	4	17
10	8	0,2	204	204	820	514	38%	12	27
10	8	0,3	307	307	717	348	52%	23	26
10	8	0,4	409	409	615	221	65%	29	19
10	8	0,5	512	512	512	137	74%	34	29

PD, el número de cadenas no propias NP, el número de cadenas no propias detectadas NPD, la tasa de fallos en la detección de hileras no propias F, el tiempo en milisegundos necesario para introducir las hileras propias en la estructura TIN y el tiempo en milisegundos que tardó la identificación de todas las hileras posibles TID.

Como se observa en los resultados de la Tabla 2, la tasa de fallos F aumenta considerablemente cuando la densidad de las cadenas propias aumenta. La cantidad de rutas generadas en los árboles de la estrutura se puebla dependiendo de la cantidad de hileras propias presentadas. Si el número de hileras propias es cercano al número total de hileras representables para un árbol específico, éste se vuelve ineficaz a la hora de detectar hileras no propias, pues encuentra caminos desde la raíz hasta las hojas para la mayoría de hileras. Por lo tanto, se recomienda el uso de un parámetro de densidad *d* bajo para la generación de los detectores.

En las Tablas 3 y 4 se muestran los resultados para una variación en el tamaño de *r*. Como se puede observar, al reducir el tamaño de *r*, la tasa de fallos F aumenta. Si el tamaño de *r* disminuye, la cantidad de árboles en la estructura de arreglos aumenta, pero también disminuye la cantidad de cadenas que cada árbol puede representar. En todas las pruebas realizadas, obtuvimos los mejores resultados cuando $r = l$, pues la cantidad de hileras representable está en su máximo. Este valor de *r* permite que el tiempo que tardamos en presentar una cadena para inserción o detección tenga complejidad $O(l)$. Sin embargo, dado que aumenta la cantidad de cadenas representables, el tamaño requerido en memoria aumenta de manera similar.

Tabla 3. Resultados para variación del tamaño de *r* para un $l = 8$ y $d = 0{,}1$

l	*r*	*d*	P	PD	NP	NPD	F	TIN	TID
8	4	0,1	25	25	231	112	52%	0	2
8	6	0,1	25	25	231	187	20%	0	2
8	8	0,1	25	25	231	191	18%	0	1

Tabla 4. Resultados para variación de r y d para un l fijo

l	*r*	*d*	P	PD	NP	NPD	F	TIN	TID
14	8	0,1	1638	1638	14746	262	99%	217	647
14	10	0,1	1638	1638	14746	7308	51%	392	663
14	10	0,2	3276	3276	13108	1969	85%	496	680
14	10	0,3	4915	4915	11469	448	97%	657	725
14	10	0,4	6553	6553	9831	94	100%	746	745
14	10	0,5	8192	8192	8192	18	100%	726	786
14	14	0,1	1638	1638	14746	12192	18%	165	339
14	14	0,2	3276	3276	13108	9103	31%	173	550
14	14	0,3	4915	4915	11469	6781	41%	434	371
14	14	0,4	6553	6553	9831	5017	49%	433	491
14	14	0,5	8192	8192	8192	3641	56%	473	511

VI. TRABAJO FUTURO

Como se mostró en la sección de complejidad espacial del algoritmo, el espacio consumido por éste se puede volver inmanejable, sobre todo cuando aumenta el tamaño del alfabeto. Surge entonces la necesidad de reducir la cantidad de espacio consumido por la estructura de árbol, aún cuando no disminuya la densidad de las hileras representadas. Al observar la Figura 4, el árbol final para a1 posee un nodo con dos hijos, que representan dos cadenas propias. Se sabe que por el número de caracteres del alfabeto utilizado (2 por ser binario), ese nodo no puede tener más hijos. Además, todos sus hijos representan cadenas propias. Es seguro decir que una vez alcanzado ese nodo, la cadena de r carácteres analizada en ese árbol será definitivamente una cadena propia, por lo que no hace falta recorrer el resto del árbol. Los nodos que se encuentren a partir de ese nodo son innecesarios. Generalizando esta idea, podemos afirmar que es posible eliminar los hijos de un nodo siempre y cuando el nodo no pueda tener más hijos (todos los elementos del alfabeto están representados en sus hijos) y todos sus hijos tengan actualmente la bandera de propiedad encendida. Si ambas condiciones son verdaderas, podemos eliminar los hijos y encender la bandera de propiedad en el nodo. Al encenderla, sería necesario revisar si el padre de este nodo también cumple con las restricciones para eliminar a sus hijos.

Otra posibilidad para la generación de detectores consiste en guardar la información de aquellas ramas que no pertenecen a la colección de hileras propias. Como parte de este cambio descartaríamos los nodos pertenecientes a las cadenas propias. En este caso sería necesario variar el algoritmo de detección, de manera que si llegamos a un nodo sin hijos, el tipo de detección sea de una cadena propia. La selección de detección por nodos de cadenas propias o nodos de cadenas no propias podría determinarse a partir de la densidad de la colección de hileras

representadas en el arreglo de árboles. Si la densidad de la colección de cadenas propias es mayor que 0,5, es muy probable que almacenar las cadenas no propias sea más conveniente. Por otro lado, sería útil utilizar distintos tipos de detección en los árboles del arreglo, según las características de cada uno de los árboles. La idea de utilizar los nodos no propios para la detección proviene de la cantidad de espacio que podría requerir representar todo la colección de cadenas propias. Es posible que junto a este cambio se pueda aplicar aún la reducción de espacio eliminando ramas del árbol. Sin embargo, queda como trabajo futuro determinar la viabilidad de tal integración.

También se hace indispensable realizar una comparación entre el algoritmo propuesto y los algoritmos presentados en [3, 5, 6]. De manera similar, el análisis entre los mecanismos de comparación constituye un trabajo futuro inminente. Un experimento interesante será la mezcla de generación de patrones de [3, 5, 6] con el mecanismo de detección de hileras propuesto en este artículo.

VII. CONCLUSIONES

En este artículo describimos un mecanismo de detección de cambios que utiliza cadenas propias de un conjunto de datos para generar un conjunto de detectores y detectar si una hilera de caracteres es propia o no. Lo interesante del mecanismo propuesto es que podemos partir de lo que conocemos, a saber, del conjunto de datos inicial, para generar los detectores. Si el conjunto de datos cambia, es necesario generar un nuevo conjunto de detectores.

La estructura utilizada por el mecanismo propuesto es un arreglo de árboles que almacena un conjunto de detectores obtenidos a partir de la presentación de las cadenas propias a la estructura. Dependiendo de la selección del parámetro r, es posible conseguir un tiempo de inserción para toda la colección de hileras propias cercano a l y en el peor de los casos no mayor que r2. Dada la similitud entre el algoritmo de inserción de cadenas y el algoritmo de detección, es posible conseguir un tiempo de ejecución en la detección cercano a l. Obtuvimos los mejores resultados de eficacia en la detección de hileras no propias cuando r = l.

REFERENCIAS

[1] Aickelin, U. y Dasgupta, D., 2005. Artificial Immune Systems. En: Search Methodology: Introductory Tutorials in Optimisation and Decision Support Techniques. New York: Springer Science + Business Media, Inc., pp. 375-399.

[2] Kim, J.; Bentley, P.; Aickelin, U.; Greensmith, J.; Tedesco, G. y Twycross, J., 2007. Immune System Approaches to Intrusion Detection - A review. En: Natural Computing, Vol. 6 (4), pp. 413-466.

[3] Ayara, M.; Timmis, J.; Lemos, L.; Castro, R. y Duncan, R., 2002. Negative Selection: How to generate detectors. En: Proceedings of the First International Conference on Artificial Immune Systems, pp 89-98.

[4] Dasgupta, D. y Atto-Okine, N., 1997. Immunity-Based Systems: A Survey. En: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Vol. 1, pp 369-374.

[5] D'haeseleer, P.; Forrest, S. y Helman, P., 1996. An Immunological Approach to Change Detection: Algorithms, Analysis and Implications. En: Proceedings of the IEEE Symposium on Security and Privacy, pp 110-119.

[6] Forrest, S.; Perelson, A.; Allen, L. y Cherukuri, R., 1994. Self-Nonself Discrimination in a Computer. En: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp 202-212.

[7] Ilgun, K.; Kemmerer, R. y Porras, P., 1995 State Transition Analysis: A Rule-Based Intrusion Detection Approach. En: IEEE Transactions on Software Engineering. Vol. 21 (3), pp 181-199.