DETECCIÓN Y MITIGACIÓN DE ANOMALÍAS EN UN FIREWALL DE RED

ANDRÉS FELIPE OCAMPO PALACIO

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE TELECOMUNICACIONES
MEDELLÍN
2016

DETECCIÓN Y MITIGACIÓN DE ANOMALÍAS EN UN FIREWALL DE RED

ANDRÉS FELIPE OCAMPO PALACIO

Trabajo de grado para optar al título de Magister en Ingeniería de Telecomunicaciones

Asesor

NATALIA GAVIRIA GÓMEZ, PhD

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE TELECOMUNICACIONES
MEDELLÍN
2016

"A mis padres y mi novia, sin su apoyo nada de esto hubiese sido posible..."

Agradecimientos

Quiero reconocer y extender mi más sentida gratitud a las siguientes personas que han hecho posible el desarrollo de esta tesis:

En primer lugar, a la Profesora Natalia Gaviria Gómez, PhD. Aceptó recibirme como su estudiante sin reparos; me brindó la libertad para explorar diferentes caminos en mis investigaciones; allí su paciencia y valiosa asesoría fueron fundamentales, en especial en esos momentos donde el horizonte parecía extraviarse.

A mis amigos Gil y Edward (en tu memoria).

A mi hermosa novia, Lina. Ha sido paciente e incondicional cuando más la necesité. Quien me movió a retomar la motivación para seguir de cara a los sueños. Ella sabe que esto es un logro para ambos.

Finalmente, quiero agradecer a mis padres. Ellos comenzaron esta tesis 29 años atrás... Han sido mi ejemplo, mi guía y mi apoyo a lo largo de mi vida. Gracias infinitas.

Resumen

Para alcanzar la resiliencia en redes de telecomunicaciones es necesario el planteamiento de metodologías que persigan este objetivo desde cada una de las etapas y dispositivos de la red, proporcionando la capacidad de seguir operando a niveles aceptables de servicio ante la ocurrencia de eventos infortunados que comprometen su correcto funcionamiento. El *Firewall* de red es uno de los dispositivos más importantes dentro del esquema de seguridad y protección de una red; es concebido como la primera línea de defensa del sistema, jugando un rol crucial al proteger la red de los flujos de tráfico que ingresan.

El *Firewall* es por tanto el blanco de ataques maliciosos que buscan doblegar su capacidad de filtrar tráfico; de lograr la salida de operación de este servicio, la red estaría a merced de nuevos ataques con cosecuencias que pueden llegar a ser debastadoras. En consecuencia, es necesario el planteamiento de metodologías que provean la capacidad de sobreponerse a este tipo de eventos manteniendo niveles mínimos de operación, resiliencia.

En este trabajo se propone una metodología de dos etapas. En primer lugar, la detección temprana de anomalías en los niveles de utilización de los recursos computacionales del *Firewall*, ocasionadas quizá, por ataques *DoS* que apuntan a las últimas reglas del mecanismo de seguridad del *Firewall*, a fin de generar la saturación de los recursos computacionales hasta colapsar el servicio. Tal detección se da mediante el monitoreo en tiempo real del nivel de utilización de la *CPU*; de superar un umbral definido, se lanza una alarma que informa de la ocurrencia del evento. Para definir el umbral, se derivó un modelo teórico para el estudio de rendimiento del sistema en diferentes escenarios, haciendo posible una clasificación de comportamiento normal o atípico en la utilización de recursos.

La segunda etapa consiste en la mitigación o remediación de las anomalías. Se desarrolló un algoritmo de planificación del orden de interrogación de las reglas del *Firewall*. El problema fue formulado como un programa entero binario sobre un grafo bipartito entre el conjunto de reglas, y una entidad definida en el marco de este proyecto denominada los estados de servicio. Se trata de una particularización del problema *Maximum Weight Match*; dada la complejidad computacional de este problema, se plantea un algoritmo para la obtención de una solución aproximada mediante la adaptación del *Greedy Maximal Match Scheduling*. Los resultados obtenidos evidencian que bajo la influencia de un ataque *DoS*, se logra mitigar las posibles anomalías en la utilización de la *CPU*, retornando rápidamente a valores de comportamiento normal, y garantizando la operabilidad del sistema incluso ante la presencia de eventos infortunados.

Índice general

Índice de figuras

Introducción

La resiliencia en redes de telecomunicaciones es un concepto que abarca diversas disciplinas en procura de mantener un nivel aceptable de servicio cuando se está bajo presencia de eventos infortunados que comprometen su desempeño [1]. Tales eventos pueden presentarse debido a fallas en los elementos de la red (nodos, enlaces), errores de configuración, ráfagas de tráfico en determinados instantes de tiempo, ataques informáticos maliciosos, entre otros; todas estas situaciones generan anomalías en el funcionamiento de los elementos y las servicios de la red. La resiliencia global del sistema se logra cuando todas las etapas y dispositivos de la red, en conjunto, implementan metodologías que los hace resilientes [2].

Uno de los elementos principales en el esquema de seguridad de una red es el *Firewall*; constituye la primera línea de defensa contra los flujos de tráfico que ingresan al sistema [3]. Una arquitectura típica de un *Firewall* de red está comprendida por un conjunto de reglas, cada una configurada con un conjunto de campos que conforman el mecanismo de seguridad; los atributos de cada paquete que ingresa al sistema son comparados con los campos de cada regla, si algunos de los campos no establece coincidencia con los atributos del paquete, éste pasa a ser interrogado por la siguiente regla; así, una a una de manera secuencial, hasta que se presenta una coincidencia entre el paquete y todos los campos de la regla, en ese punto una política de permitir o denegar el acceso del paquete es tomada. Se configura una regla por defecto al final del conjunto que será revisada si no hubo coincidencia con alguna de las reglas [4], [5]. El *Linux Netfilter* es un ejemplo de implementación

de este tipo de arquitectura.

La capacidad para permitir o denegar el acceso de paquetes ha convertido al *Firewall* en el blanco de diversos ataques que buscan comprometer su funcionamiento y en consecuencia la seguridad global de la red. Es necesario, por lo tanto, establecer una metodología que proporcione la suficiente resiliencia a este importante servicio, tal que mantenga su operación incluso bajo la incidencia de esta clase eventos.

El ataque de Denegación de Servicio (*DoS*) es el principal ataque orquestado en contra de los *Firewall* de red [6]; su objetivo está basado en saturar los recursos computacionales del servidor que alberga este servicio mediante flujos de tráfico de baja tasa configurados para establecer coincidencia con una de las últimas reglas del conjunto; la revisión de cada regla de manera secuencial hasta una coincidencia, implica un aumento en la utilización de los recursos computacionales (anomalía) que es directamente proporcional a la ubicación de la regla objeto de ataque dentro del conjunto; tal utilización puede alcanzar rapidamente niveles de saturación hasta el inminente colapso del sistema.

Una metodología de resiliencia para este servicio debe estar conformada, por lo menos, por una etapa de detección de la anomalía en la utilización de los recursos computacionales del dispositivo, y una de remediación que permita mantener la operación del sistema; posteriormente, es esencial la inclusión de una etapa de identificación del origen de la anomalía. Diferentes propuestas evidenciadas en la literatura, abordan el problema de optimizar el proceso de interrogación de las reglas [7],[8],[9]. Sin embargo, no se presenta una iniciativa de solución con miras a la resiliencia de este dispositivo [10].

En este trabajo se propone una metodología de resiliencia para un *Firewall* de red, conformada por una etapa de detección de anomalías y una de remediación. En primera

instancia, se tomó como base la utilización de la *CPU* como la métrica que da cuenta de la utilización de los recursos computacionales del dispositivo en la ejecución de una tarea en particular, el *Firewall* en este caso. A través del monitoreo de esta métrica se dispara una alerta que da cuenta de la ocurrencia de una anomalía, una vez se alcanza un umbral de utilización definido. Para el caso de remediación de anomalías, se plantea un algoritmo para la asignación del orden de verificación de las reglas, con el objetivo de minimizar la utilización de este recurso. A través de un modelamiento del *Firewall* como un sistema de encolamiento, se estableció una formulación como un problema de programación líneal entero, solucionado a través de una heurística que exhibe buenos resultados, dada la complejidad algorítmica del problema original.

Capítulo 1

Marco Teórico y Antecedentes

"La idea es tratar de dar toda la información para ayudar a los demás a juzgar el valor de tus contribuciones; no sólo la información que lleva a juzgar en una u otra dirección en particular."
— Richard P. Feynman

1.1. Hacia la resiliencia en las redes de comunicaciones

El concepto de resiliencia constituye una alternativa que busca blindar las redes de comunicaciones ante los posibles eventos que pueden afrontar como fallas, inundaciones de tráfico, rendimiento, seguridad, entre otros [11]. Comúnmente son mitigados a través de estrategias que se presentan en diferentes etapas de la red, y según cada caso; por ejemplo, desde las fases de diseño mediante la implementación de enlaces redundantes que soporten el tráfico ante eventos de falla; o en la fase de gestión y mantenimiento a través del monitoreo del tráfico; o de la implementación de *Middleboxes* con servicios como detección de intrusos (*IDS*) o *Firewalls* en aspectos de seguridad. No obstante, responden a planteamientos individuales basados en criterios subjetivos, sin seguir esquemas unificados.

Definida como la capacidad que tiene una red para defenderse y mantener un nivel aceptable de servicio ante la presencia de eventos que comprometan su correcto funcionamiento, la resiliencia en

redes representa una alternativa metodológica para conseguir sistemas robustos capaces de responder a los posibles desafíos. Debe establecerse entonces como una propiedad fundamental e integral de las redes, pensada desde la definición y planificación del sistema, y con un plan de mantenimiento y de remediación.

En el trabajo presentado en [12] se propone un marco conceptual de referencia a través de un esquema de clasificación de las disciplinas que definen las características de una red resiliente. Por una lado se tienen las disciplinas que le proporcionan a la red la capacidad de tolerar los posibles desafíos que puede afrontar el sistema: supervivencia, incluye las estrategias que buscan proveer la capacidad de soportar posibles fallas (en enlaces, en los nodos, entre otros); tolerancia a las interrupciones, brindan la capacidad de mantener la operación del sistema ante interrupciones en la conectividad de sus los componentes; y la tolerancia al tráfico, cuyos mecanismos proporciona la habilidad de soportar anomalías en los flujos de tráfico que atraviesan la red. Por otra parte, en la segunda clasificación se circunscriben las disciplinas que le permiten operar de una manera confiable (fiabilidad, seguridad y rendimiento).

Una metodogía para alcanzar la resiliencia en una red debe estar constituída, por lo menos, de una etapa de detección de los eventos que comprometan su funcionamiento, una de mitigiación que permita mantener niveles mínimos de operación, y de remediación o recuperación para retornar a los parámetros normales de red. La resiliencia global del sistema puede conseguirse implementando esta metodología en todas las fases (acceso, transporte, entre otros) y dispositivos de la red, desde su diseño y planificación, como una propiedad proactiva que se mantenga en la operación, administración y mantenimiento.

En este trabajo los desafíos o eventos infortunados que puede afrontar una red y que comprometen su correcto funcionamiento, serán tratados como anomalías o patrones atípicos en los dispositivos. Particularmente, se centra en el desarrollo de una metodología para la resiliencia en sistemas de *Firewall* de red basados en reglas, donde ataques especializados de denegación de servicio (*DoS*) son orquestados a fin de saturar los recursos de procesamiento del servidor que alberga este servicio.

1.2. Anomalías en los elementos de la red: resultado de los eventos que comprometen su funcionamiento

Una anomalía puede enterderse como un patrón de comportamiento atípico o contrario a lo que se conoce como el comportamiento normal de un sistema. Su estudio ha ganado gran importancia en el desarrollo de técnicas de detección en diversos campos de aplicación, especialmente en aquellos donde su influencia puede traer consecuencias dramáticas (detección de anomalías médicas, detección de fraudes, procesamiento de imagenes, detección de intrusos, detección de daños industriales, detección de anomalías en redes de comunicaciones). En [13] se realiza un estudio general para la detección de anomalias, sus aplicaciones, técnicas utilizadas y principales herramientas. Se presenta una clasificación para las anomalías como: puntuales, cuando una instancia individual de datos puede considerarse como anómala respecto al resto de los datos; contextuales, cuando una instancia de los datos es considerarda como anómala de acuerdo a algunos atributos únicamente [14] (tiempo, espacio, entre otros); colectivas, cuando una colección de datos relacionados son considerados como anómalos respecto al conjunto de datos completo.

Además, se establece una taxonomía de las técnicas utilizadas para la detección de anomalías en las diferentes áreas de aplicación, como se relaciona a continuación:

1. Técnicas basadas en Clasificación: un "clasificador" es entrenado a partir de un conjunto de instancias de datos a fin de derivar clases bien definidas, de acuerdo a atributos de interés. A través de pruebas para un conjunto de datos en particular, el clasificador permite categorizar los datos en alguna de las clases predefinidas [15]. Entre las herramientas matemáticas más utilizadas por esta técnica de detección están las Redes Neuronales [16], Redes Bayesianas [17], Máquinas de Soporte Vectorial [18].

2. Técnicas basadas en el análisis de vecino más cercano: se parte del supuesto de que los datos normales están distribuídos en densos vecindarios; por su parte, las anomalías se dan lejanos a tales vecindarios, posibilitando su detección.

1.3. Anomalías en los *Firewall* de red

El *Firewall* constituye uno de los principales elementos dentro del esquema de seguridad de una red de datos; su función consiste en analizar el tráfico que va desde o hacia la red, a fin de filtrar flujos que no cumplan con las políticas de seguridad establecidas. Típicamente son desplegados en el borde de la topología, de cara a internet (figura 1.1), configurando la primera línea de defensa contra flujos de tráfico que buscan atentar contra la integridad del sistema (en adelante se hará alusión a este tipo de flujos como tráfico mal intencionado). Por tal motivo, se han convertido en el objetivo de diversos ataques informáticos que buscan afectar su capacidad de filtrar tráfico con consecuencias que pueden ser castastróficas. En este trabajo se estudia el comportamiento de los *Firewall* basados en reglas, descritos a continuación, ante la presencia de anomalías que comprometen su funcionamiento.

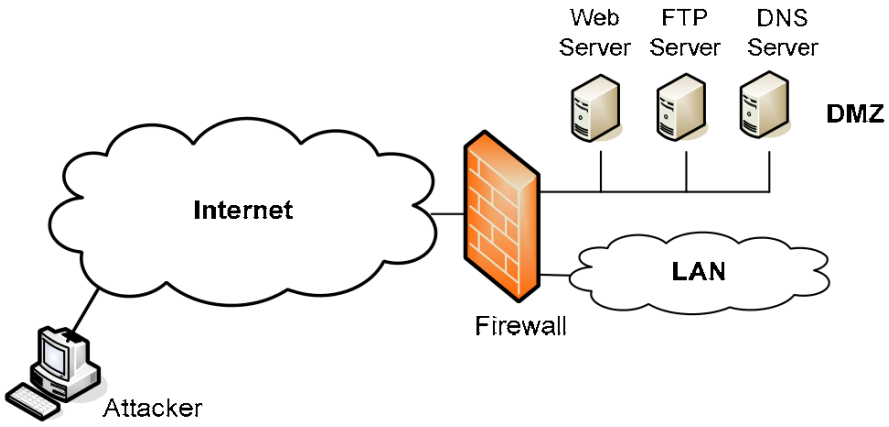


Figura 1.1: Topología de Red

Fuente: Salah, K. and Sattar, K. and Sqalli, M. and Al-Shaer, E. A probing technique for discovering last-matching rules of a network firewall. En: International Conference on Innovations in Information Technology. 2008; p.578-582.

1.3.0.1. *Firewall* de red basado en reglas

Su mecanismo de seguridad se basa en la interrogación o verificación de los flujos de tráfico a través de un conjunto o lista de reglas. Cada regla está conformada por una estructura multidimensional donde cada elemento de la estructura, denominado campo de regla, puede tratarse de una direc-

ción de origen (capa dos o capa tres), dirección de destino (capa dos o capa tres), tipo de servicio, número de puerto, número de protocolo; además, cada regla tiene asociada un campo de decisión del tipo aceptar, denegar, redireccionar un paquete [19][20].

Un paquete que ingresa al sistema pasa a ser interrogado por el conjunto de reglas de forma secuencial. Comenzando en la primera regla, sus campos son verificados uno a uno; si un evento de no coincidencia entre un campo de la regla y el paquete ocurre, éste pasa a ser interrogado por la segunda regla. De esta forma se continua hasta que todos los campos de una regla coincidan con los campos del paquete, en cuyo caso una decisión de tipo aceptar, retransmitir, o descartar el paquete, es tomada de acuerdo a las políticas de seguridad establecidas. En el caso en el que ninguna de las reglas del conjunto establezca una coincidencia con el paquete interrogado, una última regla, "la regla por defecto", con una política de decisión asociada de denegación, procesa el paquete [21].

Arquitecturas de *Firewalls* como el Linux Netfilter, FreeBSD ipfw, son ejemplos de este tipo de implementaciones, donde el conjunto de reglas del mecanismo de seguridad se define en una lista de control de acceso (ACL, de sus siglas en inglés) *Firewall*[22] [23]. Este trabajo, particularmente, está basado en la implementación del *Firewall* Linux Netfilter; la figura 1.2 brinda una ilustración de esta arquitectura; los paquetes que ingresan al sistema a través de la interfaz de red (Rx NIC) son copiados en el buffer de recepción ubicado en la memoria del kernel (Rx DMA Ring), por medio de la memoria de acceso directo (DMA) [24], [25]; el controlador del dispositivo invoca al kernel para el procesamiento de los paquetes encolados, a fin de extraer la información necesaria del paquete previa interrogación con el conjunto de reglas [26]; incluye procesamiento de capa de enlace (capa 2), procesamiento de capa de red (capa 3), verificación de errores en los encabezados, desencapsulación del paquete [27],[28].

El papel fundamental que cumplen los *Firewall* en la defensa de las redes los han convertido en blanco de ataques de denegación de servicio, *DoS*, especialmente configurados y ejecutados con el fin generar amomalías en su desempeño, mediante el consumo excesivo de los recursos computacionales del dispositivos hasta colapsar sus capacidades de procesamiento e inhabilitar el servicio. A continuación se presenta una contextualización de dichos ataques, como son orquestados para afectar a este tipo de dispositivo, el impacto que generan y las alternativas para detectarlos.

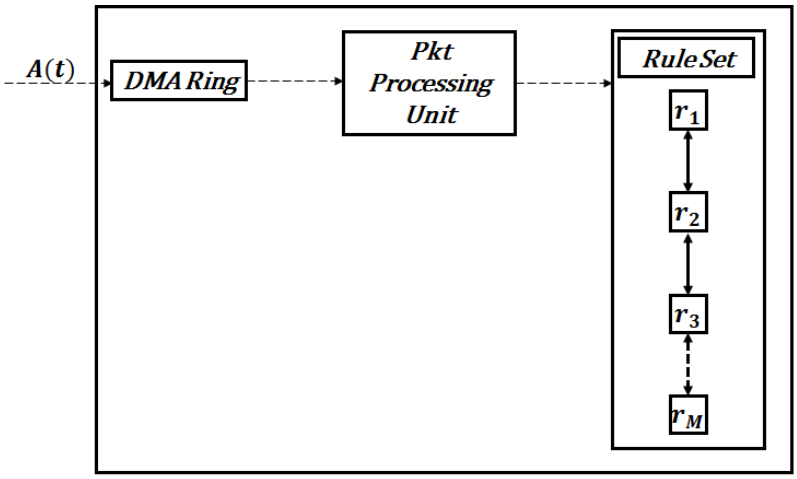


Figura 1.2: Arquitectura Linux Netfilter

1.3.1. Anomalías ocasionadas por ataques *DDoS* de baja tasa sobre *Firewall* de red

1.3.1.1. Ataques DoS

Un ataque de denegación de servicio (*DoS*) se define como una acción o conjunto de acciones ejecutadas para inhabilitar un servicio o recurso de una red [29] [30]. Estos ataques pueden llevarse a cabo mediante la saturación de los recursos del sistema que se quiere afectar, o alterando o destruyendo su información de configuración, o destruyendo o alterando sus componentes físicos [31]. El primer tipo, ataques de saturación objeto de estudio en este trabajo, se clasifican en ataques de vulneración o semántica y ataques de inundación [32]. Los ataques *DoS* de vulneración usufructan errores en el software del sistema o problemas en las políticas de seguridad para bloquear el acceso de usuarios legítimos a los servicios. Por ejemplo, al explotar los errores de desbordamiento de búfer en las rutinas de comprobación de contraseñas [33], a nivel de dispositivos de red. En cuanto a sistemas operativos, los ataques son ejecutados para explotar las vulnerabilidades que resultan por la implementación los protocolos de red; es el caso del ataque "*ping* de la muerte" donde peticiones de eco *ICMP* (ping) con tamaño de datos mayor que el tamaño máximo de un paquete *IP* según el estándar, son enviados de forma segmentada al dispositivo objetivo quien los almacena y reensambla; eventualmente el sistema operativo falla en la asignación de memoria para los segmentos de reensamble de paquete, ocasionado un desbordamiento del búfer de datos [34]. A nivel de aplica-

ciones, errores de software en alguna de las aplicaciones de red que corren en el dispositivo objeto de ataque son utilizados para consumir sus recursos e inhabilitar un servicio; por ejemplo, en el ataque conocido "*finger boom*", un usuario malicioso puede generar una rutina para una aplicación dada y que se ejecute recursivamente hasta consumir los recursos computacionales del elemento[35].

Los ataques DoS de inundación buscan agotar un recurso clave del dispositivo objetivo (ancho de banda, CPU, entre otros), a través un flujo masivo de tráfico con peticiones de servicio aparentemente legítimas, lo que supone una gran cantidad de datos a ser procesados ocasionando eventualmente la saturación de los recursos computacionales y posterior inhabilitación de un servicio dado. Por ejemplo, en ataques de inundación *UDP* un excesivo número de segmentos *UDP* dirigidos a puertos aleatorios en el dispositivo objetivo, son enviados a fin de saturar su ancho de banda afectando seriamente su conectividad con otros elementos de la red [36].

1.3.1.2. Ataques DDoS

Típicamente los ataques *DoS* son ejecutados desde una única máquina que genera el flujo de tráfico en contra del dispositivo objetivo. Otra alternativa consiste en establecer un ataque *DoS* distribuido, donde varias máquinas desde el exterior de la red generan el flujo de tráfico malicioso apuntando a mismo objetivo. La arquitectura básica de esta forma de ejecución del ataque, se basa en posicionar una cantidad determinada de estaciones, denominadas agentes, generadoras del tráfico malicioso, y un programa, conocido como coordinador (puede estar albergado de forma distribuida en varios servidores), que le indica a los agentes cuando atacar, qué atacar y cómo atacar [37]. Esta forma de ejecución del ataque, por lo general, es más poderosa que un ataque *DoS* aislado dado que el consumo de recursos en el dispositivo objetivo, responde a los flujos de tráfico individuales provenientes de las entidades que lanzan el ataque. Asimismo, defender la red de un ataque *DoS* es una labor mucho más compleja debido a los múltiples subflujos de tráfico que deben ser identificados.

1.3.1.3. Ataques DDoS de baja tasa contra *Firewall* de red

Al llevar a cabo un ataque de inundación distribuido, donde cada uno de los agentes del ataque emite un flujo de tráfico malicioso con tasa de paquetes relativamente pequeña a fin de evitar ser

detectados por sistemas *IDS*, y donde los encabezados de cada uno de los paquetes están configurados tal que establezcan un evento de coincidencia con los campos de una de las reglas ubicada al final o al fondo del conjunto de reglas del *Firewall*, y a las que se hará alusión como últimas reglas de coincidencia. De esta forma, además del procesamiento de red realizado por el kernel, la verificación de los campos de regla, secuencialmente, una a una hasta la regla de coincidencia a la que apunta el ataque y para cada paquete del flujo de tráfico malicioso, supune la ejecución de un gran número de instrucciones por parte de la *CPU* del dispositivo que alberga el servicio de *Firewall*, ocasionando anomalías en el consumo este recurso hasta niveles que pueden ocasionar un eventual colapso del servicio. Típicamente, un *Firewall* de una red coorporativa puede contener alrededor de 20.000 a 50.000 reglas [38]; tanto más cerca del final del conjunto de reglas se encuentre aquella a la que apunta el ataque, mayor será su impacto, siendo el peor caso un ataque dirigido a la regla por defecto.

1.3.1.4. ¿Cómo puede un atacante descubrir los campos de las últimas reglas de coincidencia?

Conociendo la política de seguridad de la red, con información proveniente desde su interior. También es posible determinarlas desde fuera de la red: en [39] se presenta una técnica para descubrir remotamente las últimas reglas de coincidencia del *Firewall*, consiste en enviar un tren de paquetes de "sondeo" con encabezados de paquete rudimentarios que siguen indicios de la semántica de la política de seguridad del *Firewall*, seguido de una petición a un servicio en la "zona desmilitarizada" de la red - la "zona desmilitarizada está conformada por servicios accesibles públicamente como servidores Web, email, DNS, entre otros - que busca, através de los tiempos de respuesta de dicha petición, determinar las reglas de última coincidencia; en [6] se estudia la efectividad de esta técnica mediante un diseño experimental. así mismo se analiza el impacto ocasionado en el *Firewall* cuando se sometido a un ataque *DoS* de baja tasa apuntando a una de las últimas reglas de coincidencia que se supone han sido descubiertas con la técnica mencionada. La metodología planteada en [40] busca establecer la política de seguridad del *Firewall* a través de un sistema de tres fases que comprende la generación de paquetes de prueba para verificar coincidencias con los campos de las reglas, seguido de un mecanísmo que informa cuales paquetes fueron aceptados o

denegados, y finalmente una fase de reconstrucción de la política de seguridad basada en la retroalimentación de los paquetes de prueba.

En [41] se realiza un estudio del impacto ocasionado en el *Firewall* cuando es sometido a un ataque *DoS* de baja tasa apuntando a una de las últimas reglas de coincidencia. Los resultados fueron obtenidos para flujos de tráfico malicioso configurados con diferentes tasas de paquete y apuntando a diferentes reglas. La utilización o consumo de la *CPU* del dispositivo evidencia anomalías en sus registros, llegando al punto de saturación rápidamente cuando el ataque apunta a una de las últimas reglas de coincidencia incluso con tasa de paquetes relativamente baja; en ese punto, otras métricas de rendimiento, paquetes perdidos y latencia exhiben gran degradación. Para el caso de ataques *DoS* tradicionales, el impacto sobre el *Firewall* es relevante para tasas de tráfico bastante altas.

1.3.1.5. Detección de Ataques *DoS*

El problema de detección de ataques *DoS* ha sido ampliamente estudiado, dando paso al surgimiento de metodologías y técnicas desde diferentes enfoques; en [42] se presenta una taxonomía para la clasificación de estos métodos. En primer lugar, se tienen las técnicas basadas en características; plantean la detección de los ataques mediante la identificación de una anomalía o patrón por fuera del comportamiento normal de la entidad de estudio; por ejemplo, en el trabajo presentado en [43] se define una función de entropía basada en el tamaño de los paquetes de las peticiones para los diferentes servicios en la red, tal que la detección se da a partir de las desviaciones que presentan los paquetes correspondientes a los flujos de tráfico malicioso. Otra importante clasificación incluye las metodologías basadas en el análisis de los flujos de tráfico que atraviesan la red para detectar posibles anomalías ocasionadas por un ataque; en [44] por ejemplo, a través de la extracción de características espacio-termporales empleando técnicas de correlación de las series de tiempo que describen dichos flujos, se plantea una estrategia de detección; en [45], teniendo como punto de partida el tráfico "normal" de la red con sus características en ráfaga y dependencias de largo rango (tráfico autosimilar), y a través de su descripción en series de tiempo modelado mediante *Wavelets* [46], se define una distribución de energía en el tiempo para estos flujos; se supone que las características autosimilares del tráfico "normal" hacen que la viariabilidad en dicha distribución

sea baja; en presencia de un ataque, la distribución puede presentar desviaciones en intervalos de tiempo cortos, hecho que puede ser usado para determinar la ocurrencia de evento de ataque si se supera un umbral definido. Finalmente, los métodos de detección de ataques de imitación, enmarcan las técnicas que buscan detectar ataques construídos con flujos "lícitos" desde el punto de vista de la red; el trabajo presentado en [47], basado en un método de análisis de covarianza de características extraídas de los flujos de tráfico, presenta resultados promisorios en la capacidad de detectar ataques, incluso cuando los patrones de los flujos de tráfico malicioso son similares a los del tráfico de la red.

1.4. Optimización del rendimiento de los *Firewall* de red

Varias estrategias han sido propuestas en la literatura para mejorar el rendimiento de los *Firewall* de red, mediante la optimización del proceso de interrogación del conjunto de reglas del mecanismo de seguridad:

Los autores en [48] presentan una estrategia de dos etapas; en primera instancia, se propone un algoritmo para el rechazo temprano de flujos de tráfico indeseado, basado en el análisis de las políticas de las reglas a fin de construir un subconjunto de éstas cque puedan rechazar el mayor número posible de paquetes no deseados; dada la complejidad computacional de este problema, los autores proponen una solución aproximada donde preprocesan las reglas con base en estadísticas de los flujos de tráfico, selección de reglas y rechazo temprano. La segunda fase de este trabajo consiste en la optimización del proceso de interrogación de reglas; se trata de un algoritmo para la construcción árboles que limiten la interrogación de reglas, de acuerdo a la frecuencia de coincidencia de los campos de las reglas.

En [49] se busca establecer un reordenamiento de las reglas en el *Firewall*, con base en las probabilidades de coincidencia. Se formula el problema de reordenamiento como un problema de programación líneal entero, dando solución a través de la metaheurística "*recocido simulado*". La propuesta de ordenamiento de [50] utiliza grafos acíclicos para representar las políticas del *Firewall*; posteriormente, mediante sub-grafos relacionados según políticas, coincidencias y restricciones, se busca el ordenamiento del grafo global.

Los autores de [51] plantean la formulación de un problema de programación lineal para el reordenamiento de las reglas del *Firewall*; se define en principio bloques de partición de paquetes, conformados por reglas que registran coincidencia con paquetes de configuración similar, tal que el ordenamiento se establece en subconjuntos de menor longitud y en consecuencia reduciendo la complejidad del problema. En [52] se establece un proceso de reordenamiento de las reglas del *Firewall* basado en los valores media y varianza de las estadísticas de coincidencia de las reglas. Por su parte en [53], se propone una estrategia que en lugar de interrogar las reglas del conjunto una a una, predice mediante técnicas de mineria de datos la regla más probable para establecer coincidencia, a fin de reducir el tiempo de procesamiento.

En el trabajo presentado en [54] se propone una solución heurística que consiste en reorganizar las reglas del *Firewall* de acuerdo a una función de pesos que relaciona la cantidad de coincidencia de cada una de las reglas, tal que la primer regla del conjunto será aquella que registre el mayor peso; se espera que el tiempo medio de procesamiento por paquete baje considerablemente, aumentando en consecuencia la eficiencia de este sistema. Un trabajo similar es presentado en [55], donde se definen los histogramas de coincidencia de regla, computados en ventanas de tiempo de longitud constante, comprendidas por un número fijo de segmentos en el que se procesa derterminado número de paquetes. Se propone allí un algoritmo para la reorganización de reglas, priorizando las que registre el mayor valor en el histograma de coincidencia.

Capítulo 2

Análisis de Rendimiento de un *Firewall* de Red bajo ataques *DoS*

"La ciencia es una forma de pensar, mucho más que un conjunto de conocimientos."

— Carl Sagan

Como se mencionó en el capítulo anterior, las anomalías que se estudian en el marco de este proyecto están relacionadas con el consumo excesivo de los recursos computacionales del servidor que alberga el servicio de *Firewall* basado en reglas. En este capítulo se realiza un estudio del rendimiento de este sistema cuando es sometido a flujos de tráfico normal y malicioso, a fin de tener conciencia de su comportamiento y analizar el impacto ocasionado por ataques *DoS* que buscan la salida de operación este importante componente.

2.1. Modelamiento de un *Firewall* de red como un sistema de encolamiento

Se plantea un modelo que describe el comportamiento de la arquitectura del *Firewall* Linux *net-filter* [56]. Los paquetes que arriban al sistema son recibidos a tráves de la tarjeta de red *Rx NIC*, y luego copiados a la memoria del *Kernel* utilizando la memoria de acceso dinámico *RxDMARing*. Los paquetes copiados en la memoria del *Kernel* son procesados, en primer lugar, por una etapa

que incluye procedimientos de red (verificaciones de direcciones de capa de enlace y capa de red, entre otros); luego pasan a ser interrogados por el mecanismo de seguridad del *Firewall*, conformado por un conjunto de reglas que definen el mecanismo de seguridad. Un paquete pasa a ser interrogado por la primera regla una vez el paquete que estaba siendo procesado ha abandonado el sistema, si alguno de sus atributos no coincide con alguno de los campos de la regla, el paquete pasa a ser interrogado por la siguiente regla; de esta forma se continúa hasta que se presente un evento de coincidencia entre todos los campos de la regla y los atributos del paquete; en ese punto, una decisión de eliminar o permitir el ingreso del paquete a la red es tomada [57].

Los componentes principales del *Firewall* pueden modelarse a través de un sistema de única cola y múltiples etapas de servicio secuenciales, tal y como se muestra en la figura 4.3.

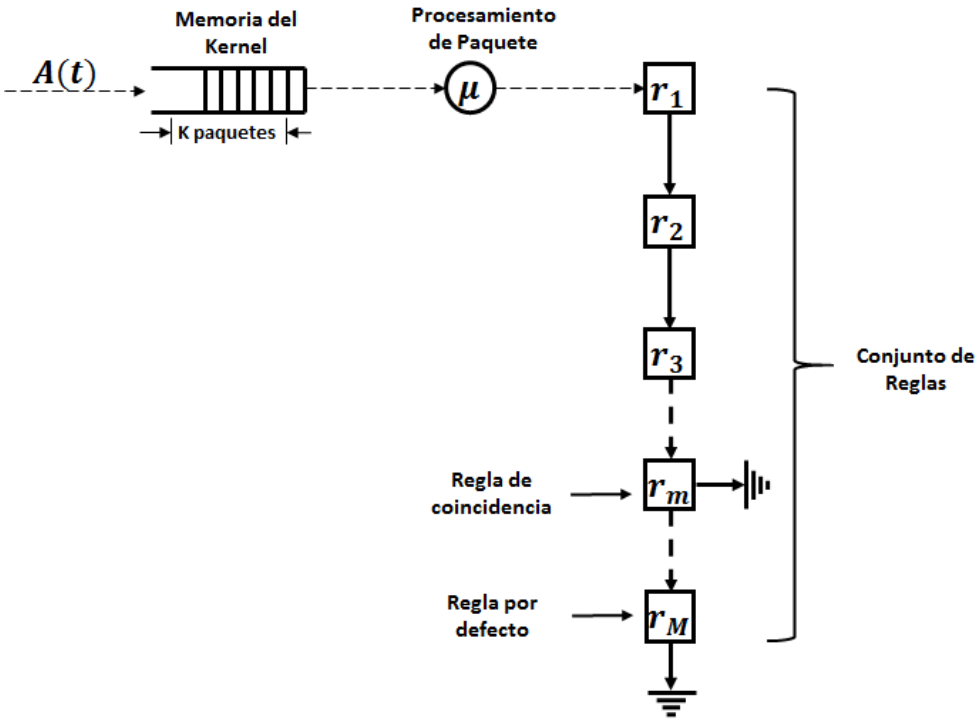


Figura 2.1: Modelo de encolamiento para el *Firewall* de red

La cola representa el búfer del *Kernel* con capacidad para albergar hasta K paquetes; el primer servidor corresponde a la unidad de procesamiento de operaciones de red efectuado por el *Kernel*;

los demás servidores $r_i, i = 1, 2, ..., M$ equivalen a las reglas del mecanismo de seguridad, en total M reglas, siendo r_M la regla por defecto. El servicio de un paquete finaliza si se da un evento de coincidencia con alguna de las reglas, dígase la regla r_m; de lo contrario, todas las reglas son procesadas y el paquete es descartado por la regla por defecto r_M.

Se asume que los procesos de llegada $A_i(t), i = 1, 2, ...$ que modelan los flujos de tráfico tanto normal como malicioso, son procesos independientes de *Poisson* de tasa λ_i paquetes/segundo(pps). Asimismo, los tiempos de servicio de los servidores del sistema se distribuyen de acuerdo a una distribución exponencial independiente, con parámetros $\mu, r_1, r_2, ..., r_m, ..., r_M$ respectivamente. En [58] y [41] se presentan los primeros en modelos de rendimiento de un *Firewall* de red como un sistema de encolamiento, suponiendo procesos de llegada de *Piosson* y tiempos de atención exponencial, cuyo tiempo medio de servicio fue calculado mediante la implementación de marcas de tiempo en cada regla dentro del *Kernel* del *Linux Netfilter*. El modelo fue validado a través de experimentación sobre una topología de red específica, los resultados evidencian que las asunciones del proceso de atención como un modelo markoviano son una buena aproximación del comportamiento del sistema.

Si bien en forma general el *Firewall* puede estar sometido a diversos flujos de tráfico, a partir de los anteriores supuestos es posible modelar los flujos de tráfico como un único proceso de llegada, de acuerdo a la propiedad de separabilidad de los procesos de *Poisson*. Así, sea $A(t)$ el proceso de llegada de paquetes con tasa $\lambda = \sum_{i=1}^{A} \lambda_i$, con A siendo el número de flujos de tráfico que inciden en el *Firewall*. De forma similar, es posible fijar los diferentes servidores que intervienen en la atención de un paquete, vistos como un único servidor, donde su tiempo de servicio es una variable aleatoria que se distribuye exponencial con parámetro $\chi = \mu + \sum_{j=1}^{m} r_i$, con m denotando la regla de coincidencia.

Estas aproximaciones permiten establecer una reducción del modelo de encolamiento original del *Firewall*, a un sistema de cola con un único servidor; ésto es factible en virtud de que las etapas de procesamiento (de red y de interrogación de reglas) que atraviesa un paquete se ejecutan una a una de forma secuencial hasta un evento de coincidencia. El servidor en este nuevo modelo computa los tiempos de procesamiento de cada una de las etapas de servicio hasta la regla de coincidencia.

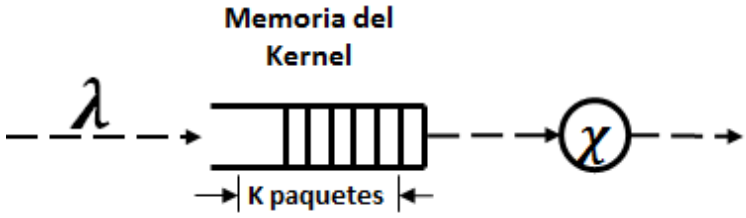


Figura 2.2: Reducción del modelo de encolamiento del *Firewall* de red

Con los supuestos en los flujos de entrada, los tiempos entre las llegadas de paquete siguen una distribución exponencial, así como el tiempo para completar el servicio de un paquete; se trata entonces de un caso típico de un sistema de encolamiento $M/M/1/K$, como se ilustra en la figura 2.2. Cabe resaltar que se ha logrado modelar el comportamiento del *Firewall* a través de un caso simple y típico en la teoría de colas, como herramienta para el análisis de rendimiento de este importante elemento en una red de datos; ésto en constraste con el trabajo presentado en [41], donde se propuso modelar un *Firewall* de red a través de una Cadena de Markov bidimensional, nociones introducidas Bianchi para el análisis de rendimiento del protocolo de acceso al medio en 801.11 [59].

La Cadena de Markov $Y(t)$ con espacio de estados $S = \{1, 2, ..., K + 1\}$ ilustrada en la figura 2.3, describe el comportamiento de este sistema. Los estados de la cadena indican el número de paquetes actualmente en el *Firewall*, incluyendo aquellos en cola y el paquete que se encuentra en servicio. $Y(t) = k$ indica que en el tiempo t el sistema se encuentra en el estado k; es decir, se tienen $k - 1$ paquetes en cola y uno en servicio.

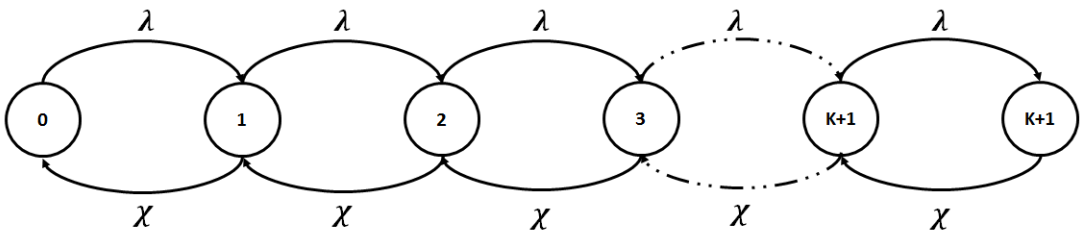


Figura 2.3: Cadena de Markov para el análisis de rendimiento del *Firewall* de red

Sea P_i, con $i \in S$ la probabilidad de estado estable. Las ecuaciones de balance para esta Cadena de Markov están dadas por:

del estado (0)

$$\begin{aligned} \lambda P_0 &= \chi P_1 \\ P_1 &= \rho P_0, \qquad \rho = \frac{\lambda}{\chi} \end{aligned} \tag{2.1}$$

del estado (1)

$$\begin{aligned} \lambda P_0 + \chi P_2 &= \lambda P_1 + \chi P_1 \\ P_2 &= \rho^2 P_0 \end{aligned} \tag{2.2}$$

del estado (2)

$$\begin{aligned} \lambda P_1 + \chi P_3 &= \lambda P_2 + \chi P_2 \\ P_3 &= \rho^3 P_0 \end{aligned} \tag{2.3}$$

del estado (K + 1)

$$\begin{aligned} \lambda P_K &= \chi P_{K+1} \\ P_{K+1} &= \rho^{K+1} P_0 \end{aligned} \tag{2.4}$$

En general, del estado (i)

$$P_i = \rho^i P_0 \tag{2.5}$$

La probabilidad total del sistema está dado por:

$$\begin{aligned} P_0 + P_1 + P_2 + ... + P_{K+1} &= 1 \\ P_0 + \rho P_0 + \rho^2 P_0 + ... + \rho^{K+1} P_0 &= 1 \\ P_0 \frac{1-\rho^{K+2}}{1-\rho} &= 1 \\ P_0 &= \frac{1-\rho}{1-\rho^{K+2}} \end{aligned} \tag{2.6}$$

A partir de la descripción de las ecuaciones de balance de esta Cadena de Markov, es posible derivar métricas que permitan analizar el desempeño del *Firewall*.

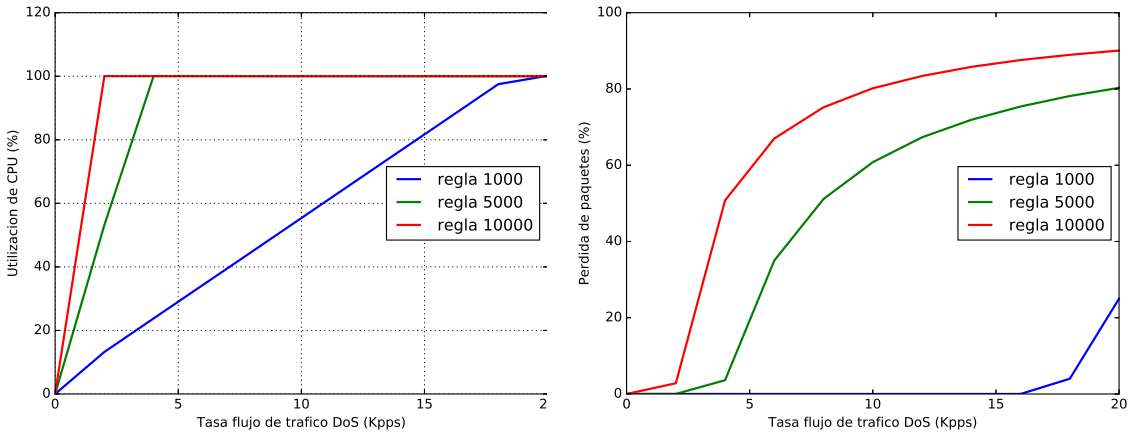
El tráfico cursado del sistema (*throughput*) está dado por la probabilidad de que la Cadena de Markov esté en el estado $Y(t) \geq 0$ y la tasa de atención total del sistema; en otras palabras, todos los paquetes que ingresan al sistema terminan atención, así:

$$\begin{aligned} \nu &= \chi \left(P_1 + P_2 + ... + P_K + P_{K+1} \right) \\ &= \chi \left(\rho P_0 + \rho^2 P_0 + ... + \rho^K P_0 + \rho^{K+1} P_0 \right) \\ &= \chi P_0 \left(\rho + \rho^2 + ... + \rho^K + \rho^{K+1} \right) \\ &= \chi P_0 \frac{\left(\rho - \rho^{K+2} \right)}{1-\rho} \end{aligned} \tag{2.7}$$

Así, la utilización promedio de la *CPU* está dada por la relación entre el tráfico cursado y el tiempo medio de servicio hasta la regla de coincidencia $\overline{X} = \frac{1}{\mu} + \frac{1}{\sum_{i=1}^{m} r_i} = 1/\chi$:

$$\begin{aligned} CPU_{util} &= \overline{X} \nu \\ &= \overline{X} \chi P_0 \frac{\left(\rho - \rho^{K+2} \right)}{1-\rho} \\ &= \frac{\rho - \rho^{K+2}}{1-\rho^{K+2}} \end{aligned} \tag{2.8}$$

Este modelo de rendimiento permite estudiar el desempeño del *Firewall* bajo diferentes escenarios



(a) Utilización de CPU bajo ataque de DoS

(b) Porcentaje de paquetes perdidos bajo ataque DoS

Figura 2.4: Impacto del ataque *DoS* apuntando a diferentes reglas

de tráfico de entrada. Interesa especialmente tener conciencia del impacto ocasionado por ataques *DoS* apuntando a las últimas reglas con el objetivo de saturar los recursos computacionales.

Con el objetivo de validar el modelo analítico, se configura un escenario de prueba que consta de un flujo de tráfico normal con coincidencia en la primera regla del conjunto a una tasa fija de 10000 *pps*. Se tomaron registros para tres ataques apuntando a las reglas 1000,5000 y 10000 respectivamente; para cada ataque se hicieron variaciones en la tasa de llegada de paquetes en múltiplos de dos, desde los 2000*pps* hasta los 20000*pps*.

La figura 2.4a muestra el comportamiento de ocupación de los recursos del *Firewall* en términos de la utilización de la *CPU* para el escenario configurado, según se definió en la ecuación 2.8. Por su parte, la figura 2.4b da cuenta del registro del porcentaje de paquetes perdidos que arriban al *Firewall*. En ambos casos se registra el valor de estas métricas para los diferentes ataques *DoS* mencionados arriba, bajo la premisa de que además el *Firewall* tendrá incidencia del flujo de tráfico normal; ésto con el objetivo de estudiar las anomalías en la utilización de los recursos ocasionadas por los ataques.

Para un *Firewall* con 10000 a 20000 reglas configuradas, un ataque apuntando a la regla 1000 no

representa una amenza que comprometa la disponibilidad de los recursos. Como puede verse en la figura 2.4a, es necesario que la tasa de paquetes del ataque sea muy alta para que la ocupación de los recursos alcance un valor significativo; a partir de los 10000 *pps* se percibe un valor de utilización de la *CPU* pon encima del 50 %; a 18000 *pps* se obtiene nivel de saturación en la utilización de los recursos (100 % de utilización de la *CPU*), lo que ocasionaría un colapso inminente. De hecho, como puede evidenciarse en la figura 2.4b, el porcentaje de paquetes perdidos pasa de 0 % hasta un 25 % a altas tasas del flujo de tráfico malicioso. No obstante, se espera que otros dispositivos complementarios de la estructura de seguridad de la red sean capaces de identificar y bloquear un flujo de tráfico malicioso ejecutado con tasas de paquetes tan altas [60].

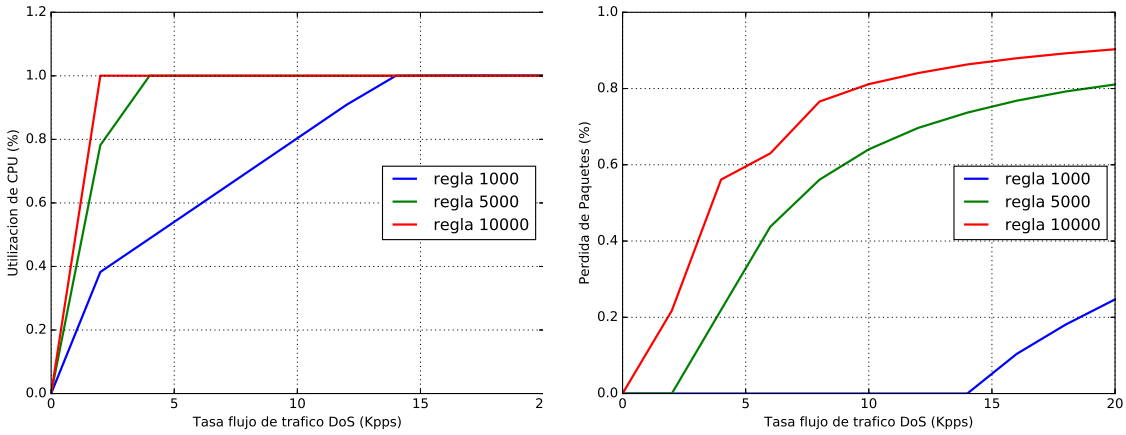
Un ataque configurado para establecer coincidencia con la regla 5000 logra saturar los recursos computacionales del *Firewall* a tasas de paquetes relativamente bajas, tal y como puede evidenciarse en la figura 2.4a; a una tasa de 4000 *pps* se alcanza una utilización de *CPU* del 100 %. A los 6000 *pps*, y con estos niveles de saturación, el búfer del *Kernel* mantiene una ocupación a tope, tal que el 40 % (figura 2.4b) de los paquetes se pierde; a una tasa de 16000 *pps* el tráfico perdido alcanza un indice del 80 %.

El caso más extremo se presenta para un ataque apuntando a la regla 10000; con un flujo de tráfico de tan solo 2000 *pps* el *Firewall*alcanza una utilización de *CPU* del 100 %; a los 4000 *pps* el tráfico perdido debido a los niveles de saturación, supera el 50 %. Así, puede evidenciarse que un ataque *DoS* de baja tasa apuntando a una de las últimas reglas del mecanismo de seguridad, genera anomalías en la utilización de la *CPU* del dispositivo, con consecuencias que pueden ser devastadores para la red. Un ataque *DoS* con tasa entre los 2000 *pps* a 6000 *pps*, puede considerarse de baja tasa, de acuerdo a la caracterización de este tipo de ataques realizada en [61], [62].

En el experimento anterior, la finalidad de fijar el flujo de tráfico normal a una tasa de 10000 *pps* y con coincidencia siempre con la primera regla es doble: en primer lugar emular el comportamiento de flujos de tráfico normal, los cuales establecen coincidencia con las primeras reglas del *Firewall*, mientras se analiza el impacto de un ataque *DoS* apuntando a una de las últimas reglas del conjunto y a diferentes tasas de paquetes. Por otra parte, la configuración del experimento sigue el escenario de prueba reportado en [41], con el fin de demostrar que el modelo de rendimiento

del *Firewall* derivado arriba, logra resultados similares a los reportados en dicho trabajo, con un modelo teórico mucho más simple en términos del tratamiento matemático.

Sin embargo, un segundo escenario donde un flujo de tráfico normal a una tasa fija de 10000 *pps* puede hacer coincidencia con reglas en el intervalo $[1-1000]$ es evaluado; un caso más realista respecto a las coincidencias que típicamente establecen los paquetes del tráfico normal con las reglas del *Firewall*[6]. Cada paquete de este flujo hace coincidencia con una de las reglas en dicho intervalo, aleatoriamente siguiendo una distribución uniforme. Similar a la configuración del experimento anterior, se ejecuraron tres ataques independientes apuntando a las reglas 1000, 5000 y 10000, y variando en cada caso la tasa de paquetes desde los 2000 *pps* hasta los 20000 *pps*, en múltiplos de dos.



(a) Utilización de CPU bajo ataque de DoS (b) Porcentaje de paquetes perdidos bajo ataque DoS

Figura 2.5: Impacto del ataque *DoS* apuntando a diferentes reglas

Los resultados obtenidos para este escenario donde el flujo de tráfico normal hace coincidencia con las reglas ubicadas en el intervalo $1-1000$, a diferencia del caso anterior donde el tráfico normal fue configurado para establecer coincidencia únicamente con la regla 1 (figura 2.4), son reportados en la figura 2.5.

Para un ataque apuntando a la regla 1000 a una tasa de 2000 *pps* la utilización de la *CPU* alcanzó el 30 %, en comparación con el mismo ataque en el escenario anterior donde la utilización de

la *CPU* fue del 10 %; a 4000 *pps* esta misma métrica superó el 40 %, mientras que en el caso anterior el reporte fue de 20 %; con 10000 *pps* la utilización de *CPU* percibe un valor crítico de 80 %; a los 14000 *pps* la utilización de *CPU* alcanza el nivel de saturación de 100 %; su contrapartida del escenario anterior alcanzó tal nivel de saturación hasta una tasa de 18000 *pps*. El porcentaje de paquetes perdios alcanza un valor de 10 % para una ataque de tasa 16000 *pps*, de 20 % para el caso de 18000 *pps*, y de 25 % para un ataque de 20000 *pps*. En el escenario anterior, con un ataque de 18000 *pps* comienza a percibirse perdida de paquetes con un valor de 10 %; a 20000 *pps* el porcentaje de paquetes perdios fue de 25 %.

Para ataques apuntando a la regla 5000, a una tasa 2000 *pps* se origina una utilización de la *CPU* del 60 % aproximadamente, en comparación con el escenario anterior que registró un valor de 50 %; para un ataque de 4000 *pps* se percibe un valor de esta métrica al nivel de saturación de 100 %, con un porcentaje de paquetes perdidos de 20 %; su contrapartida en el escenario anterior alcanza también el nivel de saturación pero con una perdida de paquetes de menos del 10 %. Para ataques con tasa de 10000 *pps*, el porcentaje de paquete perdidos supera el 60 %; ataques a partir de 18000 *pps* exhiben perdidas de paquetes de 80 %.

Para los ataques apuntando a la regla 10000, a una tasa de tan sólo 2000 *pps* se alcanza el nivel de saturación de 100 %, igual que el escenario anterior. La diferencia sustancial se presenta en la estadística de porcentaje de perdida de paquetes; para ataques con tasa de 4000 *pps* se obtuvo un valor de 60 % de paquetes perdidos, mientras que en el caso anterior para un ataque con la misma tasa fue de 40 %. En ambos casos, para ataques con tasa desde los 10000 *pps* el porcentaje de paquetes perdidos fue de 80 %, y para 20000 *pps* el valor registrado fue de 90 %.

Puede verse que los resultados de ambos escenarios convergen a los mismos valores para ataques con alta tasa de paquetes; no obstante, para este último escenario en el que el flujo de tráfico normal hace coincidencia con las reglas en el intervalo $1 - 1000$, ataques con menor tasa presentan valores de utilización de la *CPU* más altos que los conseguidos por los mismos ataques en el escenario reportado en la figura 2.4. Este hecho es lógico, toda vez que para un flujo de tráfico que pueda hacer coincidencia con reglas en el intervalo $1 - 1000$, el *Firewall* tendrá mayor grado de procesamiento que al hacer coincidencia únicamente con la regla 1.

Capítulo 3

Detección de Anomalías en un Firewall de Red

"Las computadoras son increíblemente rápidas, precisas y estúpidas. Los seres humanos son increíblemente lentos, imprecisos y brillantes. Juntos son poderosos más allá de la imaginación."
— Yan Ayrton

Un *Firewall* de red configura un servicio esencial en el esquema de seguridad para proteger la integridad de las redes de datos. Puede ser desplegado en cualquier sistema de computo capaz de procesar las operaciones definidas para éste.

Los ataques de *DoS* en contra de *Firewall* de red buscan saturar los recursos computacionales del dispositivo que alberga este servicio, a fin de ocasionar su salida de operación con consecuencias que pueden llegar a ser debastadoras para toda la red. Si el atacante descubre una de las últimas reglas del conjunto que conforman el mecanismo de seguridad del *Firewall*, un flujo de tráfico puede ser configurado para establecer coincidencia con los campos de dicha regla. El tiempo de ejecución que tomaría la interrogación de cada una de las reglas, de forma secuencial al procesar cada uno de los paquetes del flujo de tráfico malicioso, implica una alta utilización de los recursos del sistema, hasta niveles que llevan rapidamente a su saturación, generando una anomalía de procesamiento

en comparación con los niveles que supone un comportamiento normal.

Lo anterior abre la necesidad de diseñar una metodología que permita detectar este tipo de anomalías de manera oportuna, a fin de tomar medidas de remediación que eviten el colapso del sistema y mantener el servicio operativo aún en presencia de este tipo de dasafíos. A continuación se describe una metodología de detección de anomalías en un *Firewall* de red, ocasionadas por ataques *DoS* que apuntan a las últimas reglas del mecanismo de seguridad, basada en el monitoreo de la utilización de la *CPU*.

3.1. Utilización de la CPU del Firewall de Red

Un *Firewall* de red puede ser implementado en cualquier sistema computacional capaz de procesar las tareas que supone este servicio (desencapsulamiento de paquetes, interrogación de reglas, etc). Su desempeño está sujeto a las capacidades de computo del dispositivo físico que lo alberga. Una importante métrica para el análisis del desempeño de un sistema computacional está dada por la utilización de la *CPU*; ésta da cuenta de la ocupación de los recursos del dispositivo mientras ejecuta tareas provenientes de diferentes aplicaciones o servicios [63], de gran utilidad en la arquitectura de computadores a la hora de comparar diferentes implementaciones del conjunto de instrucciones de un procesador, o para verificar el comportamiento temporal de ejecución de una aplicación determinada.

La medida de desempeño por excelencia de un sistema computacional está dada en términos del tiempo de ejecución de una programa. Es necesario diferenciar esta medida de tiempo; puede definirse como el tiempo total transcurrido hasta la culminación de determinada tarea; pero si se tiene en cuenta que el procesador puede trabajar en diversas aplicaciones en simultáneo, una medida más conveniente está dada por el valor de tiempo que toma la *CPU* en la ejecución efectiva de esa tarea sin considerar las otras operaciones ejecutadas antes de su culminación, denominada *tiempo de CPU* [64].

Dada la naturaleza discreta de los sistemas computacionales, la ejecución de las tareas de un determinado programa se presentan en intervalos de tiempo de longitud constante, denominados

ciclos del reloj; este valor es también conocido como *tasa de reloj*, corresponde al inverso de la longitud de este intervalo; a menor longitud menor será el tiempo de ejecución de las tareas. En este trabajo se deriva la métrica de utilización de la *CPU* del dispositivo en la ejecución de las tareas correspondientes al *Firewall* de red, a partir del número de intervalos o ciclos de reloj necesarios para el procesamiento de todos los estados de servicio, incluyendo las operaciones de red efectuadas por el *Kernel* a cada paquete que ingresa al sistema (procesamiento de capa de enlace y de red), hasta la interrogación de cada paquete con las reglas que conforman el mecanismo de seguridad.

Sea C_{fw}, el número de ciclos de reloj consumidos por la *CPU* en la ejecución de los estados de servicio del *Firewall*, calculados así:

$$C_{fw} = \frac{T_{cpu}^{fw}}{T} \tag{3.1}$$

Donde T_{cpu}^{fw} es el tiempo de *CPU* para el servicio del *Firewall*. Como se mencionó atrás, corresponde al tiempo consumido por la *CPU* en la ejecución de las tareas del *Firewall* específicamente; T es la duración de los ciclos de reloj de la *CPU*.

Sea C_{cpu} el número total de ciclos de *CPU*, calculado como la relación entre el tiempo total de *CPU* transcurrido durante la ejecución de las tareas, y la duración de los ciclos de reloj de la *CPU*, así:

$$C_{cpu} = \frac{T_{cpu}}{T} \tag{3.2}$$

Así, la utilización de la *CPU* para el *Firewall* puede calcularse como la relación entre el número de ciclos de reloj consumidos por los estados de servicios del *Firewall*, y el número total de ciclos transcurridos para la *CPU*, así:

$$CPU_{util}(\%) = \frac{C_{fw}}{C_{cpu}} \tag{3.3}$$

3.2. Monitoreo de la Utilización de la CPU del Firewall de Red

Se define una estrategia para el monitoreo de la utilización de la *CPU*, con el objetivo de conocer el comportamiento del dispositivo que alberga el servicio de *Firewall* de red, en cuanto al consumo de los recursos computacionales al momento de ejecutar las tareas asociadas.

La estrategia consiste en posicionar marcas (*timestamps*) que registren los tiempos de *CPU* de los diferentes estados de servicio del *Firewall*. La etiqueta *networking_processing_unit_srv_time* registra el tiempo de servicio que toma el *Kernel* en el procesamiento de red de cada uno de los paquetes admitidos al *DMARing*, tal que el tiempo de *CPU* hasta este estado de servicio se computa como:

```
firewall_cpu_time = firewall_cpu_time +
                    networking_processing_unit_srv_time
```

Por su parte, la etiqueta *rule_i_srv_time_* registra el tiempo de servicio de la regla i en la interrogación de un paquete, y el tiempo de *CPU* es computado así:

```
firewall_cpu_time = firewall_cpu_time + rule_i_srv_time
```

El tiempo de servicio de cada regla es registrado y computado hasta la ocurrencia de un evento de coincidencia. Así, el tiempo de *CPU* da cuenta del tiempo de procesamiento tomado por el acumulado de todos los estados de servicio que intervinieron hasta la regla de coincidencia.

Se ha definido dentro del *Kernel* del *Firewall* un método que permite calcular la utilización de la *CPU* del dispositivo, de acuerdo a las ecuaciones 3.1 y 3.3 definidas arriba; este valor es computado cada vez que se presenta un evento de coincidencia. De esta forma, esta métrica puede ser monitoreada en todo instante de tiempo, facilitando la detección de anomalías en la utilización de los recursos del *Firewall*, ocasionados por ataques maliciosos o flujos masivos de tráfico legítimo.

```
if coincidencia_regla_i:
```

```
firewall_cpu_util = (firewall_cpu_ciclos_reloj) / (cpu_ciclos_reloj)
```

Las etiquetas descritas están diseñadas para ser instrumentadas dentro del código del *Kernel* del *Linux-Netfilter*. En el marco de este proyecto, la instrumentación fue realizada sobre el software desarrollado para emular el comportamiento del *Firewall* de red , denominado "Módulo de Simulación de un Firewall de red para el Simulador de redes ns-2 (Network Simulator 2)", el cual ha sido detallado en el anexo A.1.

Con el objetivo de verificar el comportamiento de este componente de monitoreo de la utilización de la *CPU* en la ejecución de las tareas correspondientes al *Firewall* de red, se estabeció un marco experimental utilizando el simulador descrito en A.1. Se configuró un servidor de 2.4 Ghz de frecuencia de reloj, con capacidad de 512 paquetes en el búfer del *DMARing*; la unidad de procesamiento para las operaciones de red del *Kernel* fue configurada para atender paquetes con un tiempo de servicio medio de 2.65 μseg; se configuraron 20000 reglas dentro del mecanismo de seguridad, cada una de ellas con un tiempo medio de servicio de 0.05 μseg en la interrogación de los paquetes que ingresan al sistema; se tiene entonces un total de 20001 estados de servicio. Se efectúo una serie de experimentos descritos a continuación, en los que el *Firewall* es sometido a diferentes flujos de tráfico normal y malicioso, con tasas de paquetes dadas en unidades de paquetes por segundo (pps).

Como se mencionó anteriormente, este trabajo centra su análisis en ataques de *DoS* (tráfico malicioso) apuntando a las últimas reglas del mecanismo de seguridad de un *Firewall* basado en reglas, orquestados con el objetivo de comprometer su capacidad de filtrar el tráfico desde o hacia la red. Entendiendo además que el *Firewall* procesa flujos de tráfico de diversa naturaleza (normal, malicioso), generalmente en simultáneo, es necesario tener noción del comportamiento de este dispositivo cuando está bajo la influencia únicamente de tráfico normal, como punto de referencia que permita determinar el verdadero impacto de un ataque. De esta forma, se ejecuta un flujo de tráfico normal configurado para coincidir con las reglas en el intervalo $[1, 1000]$, aleatoriamente, siguiendo una distribución uniforme. En la figura 3.1 se evidencia la utilización de la *CPU* del *Firewall* para tres flujos de tráfico ejecutados independiente, a tasas 10000 pps, 15000 pps y 20000 pps. La utilización de la *CPU* alcanza valores estables en 25 %, 40 % y 53 % aproximadamente, para cada flujo respectivamente; incluso ante un flujo de tasa relativamente alta como el de 20000 pps, el *Firewall*

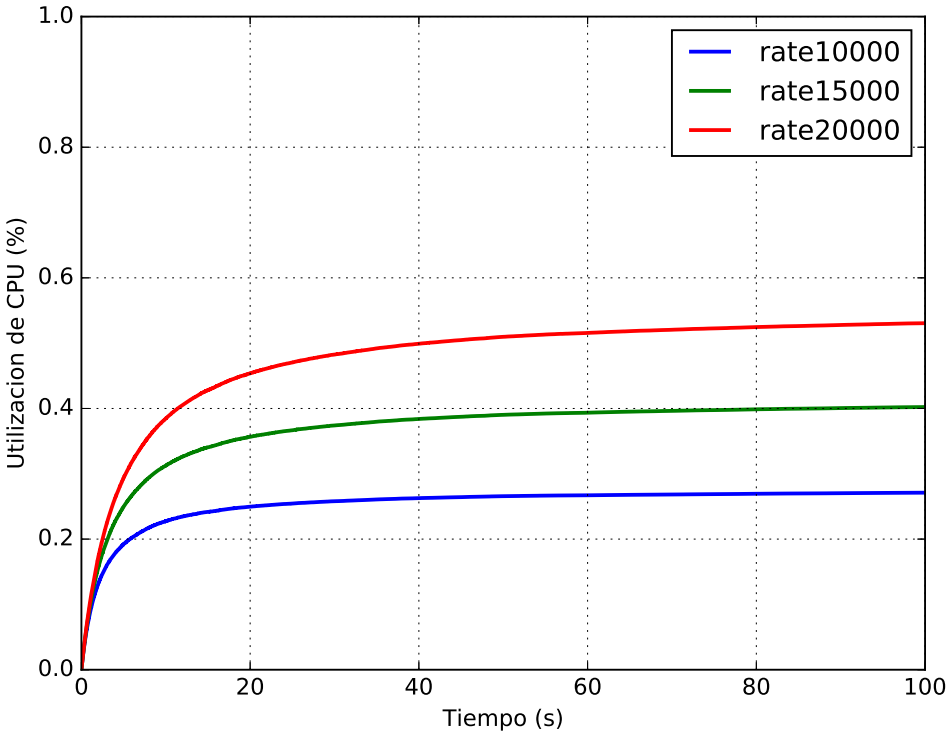


Figura 3.1: Proceso de monitoreo de utilización de la *CPU* - tráfico normal

alcanza un nivel aceptable en la utilización de los recursos (50 %) . Cabe resaltar que este tipo de tráfico puede considerarse normal, toda vez que sus paquetes pueden establecer coincidencia con diferentes reglas del mecanismo de seguridad; a diferencia de los ataques *DoS*, donde se busca hacer coincidencia con una única regla ubicada al final del conjunto.

La figura 3.2 muestra el comportamiento de la métrica de utilización de la *CPU* cuando el *Firewall* es sometido a un flujo de tráfico normal de tasa 10000 pps, configurado para establecer coincidencia con reglas ubicadas en el intervalo $[1, 1000]$, de forma aleatoria de acuerdo a una distribución uniforme; además es sometido a un flujo de tráfico malicioso o ataque *DoS* que apunta a la regla 9800, una de las últimas reglas del conjunto, a una tasa de 4000 pps; este tipo de reglas pueden ser descubiertas a través de un ataque de comprobación como se describió en el capítulo 1 [39]. La utilización de la *CPU* alcanza rápidamente un valor de saturación por encima del 90 %, debido

a la incidencia en simultáneo de ambos flujos de tráfico, y la carga de procesamiento que ello supone.

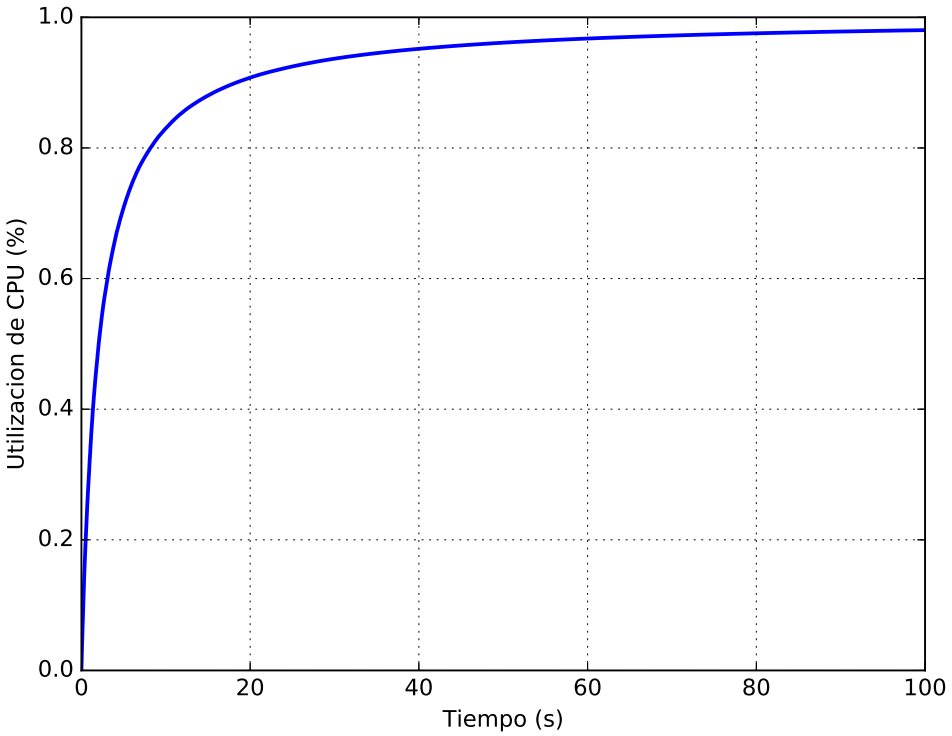


Figura 3.2: Proceso de monitoreo de utilización de la *CPU*

En un segundo experimento el *Firewall* fue sometido a un flujo de tráfico normal de tasa 10000 *pps*, donde cada paquete puede coincidir con una de las reglas ubicadas en el intervalo $[1, 1000]$, aleatoriamente de acuerdo a una distribución uniforme, entre 2 a 60 segundos desde el inicio de la observación; un segundo flujo de tráfico normal de menor tasa al inicial (5000 *pps*), incide entre los 80 a 300 segundos para hacer coincidencia con las reglas ubicadas en el intervalo $[1, 200]$, aleatoriamente de acuerdo a una distribución uniforme; además de un flujo de tráfico correspondiente a un ataque *DoS* de baja tasa que apunta a la regla 5000, y que incide en el *Firewall* a una tasa de 4000 *pps* entre los 20 a 150 segundos de la observación. En el capítulo 2 para el análisis de rendimiento de un *Firewall* de red, se mostró como flujos de tráfico malicioso a bajas tasas impactan negativamente este dispositivo; en concordancia con la caracterización de tráfico *DoS* de baja tasa

(2000 *pps* a 6000 *pps*) reportados en [62].

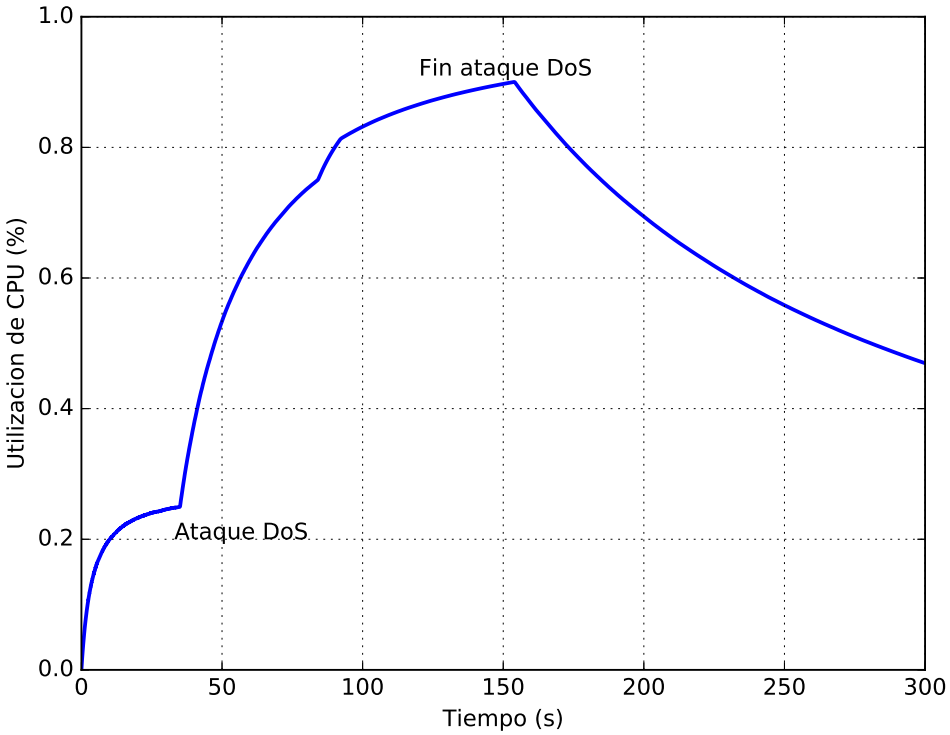


Figura 3.3: Proceso de monitoreo de utilización de la *CPU*

La figura 3.3 muestra el comportamiento de la utilización de los recursos computacionales por parte del *Firewall* en términos de la utilización de *CPU*, cuando es sometido a diferentes flujos de tráfico. Puede notarse un aumento significativo de esta métrica en el momento en el que el se ejecuta el ataque *DoS* alcanzando rápidamente un nivel de saturación. Una vez culmina el ataque, el nivel de utilización de *CPU* retorna a valores normales.

Esta metodología brinda una noción del comportamiento de la utilización de los recursos computacionales del servidor en la ejecución de las tareas correspondientes al servicio del *Firewall*. Desde el punto de vista del objetivo de este trabajo, constituye la herramienta para establecer la existencia de anomalías generadas por ataques que buscan sacar de operación este importante servicio del esquema de seguridad de una red.

3.3. Detección de Anomalías en el Firewall de Red

Con la introducción del módulo para el monitoreo de la utilización de la *CPU* descrito en la sección anterior, es posible establecer la ocurrencia de anomalías en la utilización de los recursos computacionales del servidor que alberga el servicio del *Firewall* red.

En el capítulo 2 se realizó un análisis de rendimiento de un *Firewall* de red cuando es sometido a flujos de tráfico normal y malicioso. Se evidenció como la ejecución de un ataque *DoS* de baja tasa apuntando a una de las últimas reglas del esquema de seguridad degrada drásticamente el rendimiento del servidor, hasta llegar rápidamente a un estado de saturación (más del 90 % de utilización) y potencial salida de servicio en un tiempo relativamente corto (figura 3.2). Así, un aumento súbito en el valor de esta métrica durante la ejecución de tareas relacionadas con el procesamiento del *Firewall*, es un claro indicio de una anomalía generada por tráfico malicioso.

Se establece una metodología para la detección de anomalías en un *Firewall* de red, que consiste en monitorear el nivel de procesamiento a través del valor de utilización de la *CPU*, mediante el módulo descrito en la sección 3.2. Sea U_{max} el umbral de utilización de *CPU*, por debajo de este valor se considera que el *Firewall* opera en condiciones normales y será capaz de soportar las siguientes cargas de procesamiento. En el momento en el que el valor de utilización de *CPU* supere este umbral, se considera que una anomalía de procesamiento está ocurriendo, y la saturación de los recursos es inminente. En ese punto, se dispara una alerta que notifica al *Kernel* acerca de la ocurrencia de una anomalía; aviso que puede dar paso a la implementación de estrategias para mitigar el impacto en el desempeño del *Firewall* de red.

```
if coincidencia_regla_i:
  if firewall_cpu_util > U_max:
    kernel()->anomaly()
```

La figura 3.4 representa el registro de la utilización de la *CPU* para un *Firewall* compuesto por un conjunto de 10000 reglas, con mismas configuraciones que las expresadas en la sección 3.2. En este caso se sometió al *Firewall* a dos flujos de tráfico normal y uno malicioso; el primero, correspondiente al tráfico normal con tasa de paquetes de 10 *Kpps*, con coincidencia entre las primeras

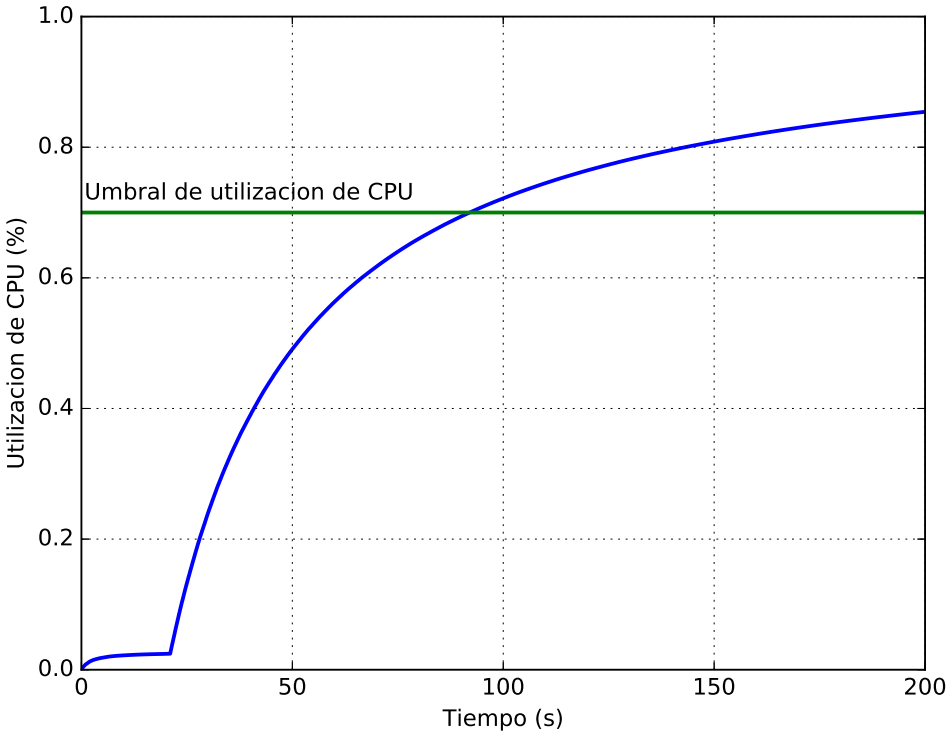


Figura 3.4: Proceso de monitoreo de utilización de la *CPU*

reglas del conjunto $1 - 1000$ aleatoriamente de acuerdo a una distribución uniforme; en el caso del flujo de tráfico configurado para el ataque *DoS*, se estableció un flujo de tráfico de tasa 4 *kpps*, apuntando a la regla 5000.

Se ha definido un valor de $U_{max} = 70\,\%$; el ataque *DoS* es ejecutado a los 30 segundos; alrededor de los 90 segundos la utilización de la *CPU* alcanza el valor umbral, en tanto el *Kernel* es notificado del evento, permitiendo la ejecución de una posible estrategia de remediación que permita soportar este evento infortunado sin la salida de operación del *Firewall*.

Cabe resaltar que este tipo de ataques *DoS* no son la única causa que puede generar anomalías en la utilización de la *CPU* del dispositivo; altas tasas de paquetes de tráfico normal o carga excesiva del procesador debido a otras tareas que se estén ejecutando, también pueden generarlas eventual-

mente. Así, este módulo está basado en determinar cuando la utilización de los recursos del *Firewall* alcanza niveles de saturación peligrosos para su operación, como indicador de la ocurrencia de la anomalía independiente de la causa. Sin embargo, como se evidenció en el análisis de rendimiento del capítulo 2, un caso crítico se presenta cuando un ataque apuntando a una de las últimas reglas del conjunto logra llevar el consumo de recursos a niveles dramáticamente altos en poco tiempo, aún con una tasa de paquetes relativamente baja.

Capítulo 4

Mitigación de Anomalías en un *Firewall* de red

"Todos los modelos están equivocados, pero algunos modelos son útiles"
— George E.P. Box (1979), En: Robustness in the strategy of scientific model building

La metodología más elemental de un sistema de red resiliente, según se mencionó en la sección 1.1, está compuesta por una etapa de detección e identificación de desafíos, y una de remediación y recuperación; en el caso de este trabajo, tales desafíos están dados en términos de anomalías en el consumo de recursos de un *Firewall*, ocasionados por potenciales ataques de *DoS* que buscan poner en peligro la seguridad de la red.

En el capítulo anterior se definió una metodología para la detección de anomalías, que consiste en el monitoreo "en línea" de una importante métrica como la utilización de la *CPU*; si este valor supera un umbral de tolerancia definido en el procesamiento de los flujos de tráfico que ingresan al sistema, se dispara una alarma que informa al *Kernel* acerca de la ocurrencia de la anomalía.

Una vez detectada la anomalía, se plantea una metodología de remediación que ayude a aliviar el impacto negativo en el consumo de los recursos computacionales que ésta ocasiona.

4.1. Estados de Servicio

Resaltando nuevamente que este trabajo está basado en la arquitectura del *Firewall* de red *Linux Netfilter*, los paquetes que ingresan al sistema son recibidos por la tarjeta de red (*Rx NIC*) y luego copiados en la memoria del *Kernel* mediante el *Rx DMA Ring*. Aquellos que han sido almacenados en este búfer son atendidos, en principio, por la unidad del kernel encargada de efectuar operaciones de red (de capa dos y tres) sobre cada paquete; posteriormente, pasan a ser atendidos por el conjunto de reglas del *Firewall* de forma secuencial, hasta que ocurre un evento de coincidencia respecto a todos los campos de una regla [65]; en ese momento se toma la decisión de dejar pasar el paquete hacia la red, o desecharlo.

El *Firewall* de red puede ser modelado entonces como un sistema de encolamiento de múltiples etapas de servicio, como se describió en el capítulo 2. La cola representa la memoria de acceso del *Kernel*, los servidores secuenciales corresponden al procesamiento de red y al procesamiento de interrogación de cada regla, hasta el evento de coincidencia; la figura 4.1 ilustra el modelo de encolamiento para el *Firewall* de red.

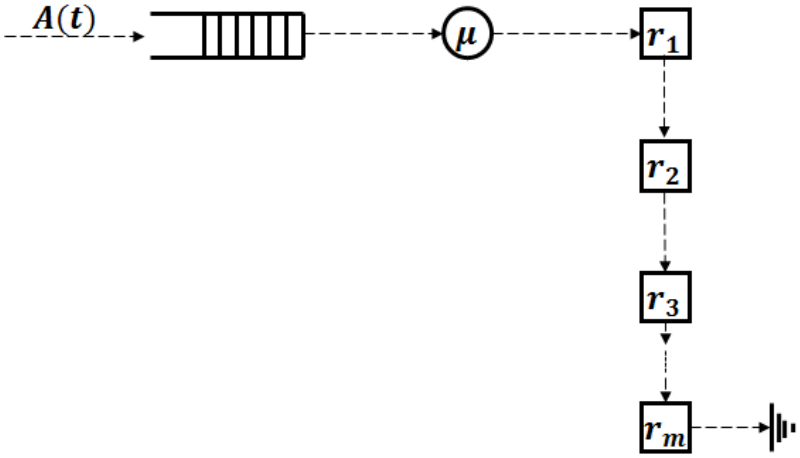


Figura 4.1: *Firewall* de red como sistema de encolamiento

Sea $R = \{r_1, r_2, ..., r_M\}$ el conjunto de reglas del mecanismo de seguridad del *Firewall*, donde $|R| = M$, es el cardinal del conjunto correspondiente al número total de reglas configuradas. Se

define el conjunto de estados de servicio de reglas $S = \{s_1, s_2, ..., s_M\}$, representa las posibles etapas de procesamiento del *Firewall* en la interrogación de las reglas. Cada elemento $s_i \in S, i = 1, 2, ..., M$ tiene asociada la regla $r_j \in R,\ j = 1, 2...M$; es decir, el estado de servicio s_i debe procesar la regla r_j; el estado s_1 indica la primera etapa de atención del *Firewall* en el conjutno de reglas, es decir la pirmera regla que debe ser procesada o interrogada; s_2 referencia la segunda regla a ser procesada; de esta forma, el estado de servicio s_M indica la última regla que debe ser interrogada. Para la implementación inicial del *Firewall* las reglas son ejecutadas de forma secuencial, desde la primera regla del conjutno r_1, hasta la regla de coincidencia r_m, tal que $i = j$; en otras palabras, en el primer estado de servicio s_1 se ejecuta la primera regla del conjunto r_1, y de esta forma respectivamente hasta el evento de coincidencia, donde en el estado de servicio s_m se interroga a la regla r_m. La figura 4.2 ilustra la correspondencia entre los estados de servicio y las reglas que en cada uno de ellos se debe procesar, para cuando el *Firewall* de red sigue su implementación original (sin ninguna estrategia de remediación de anomalías).

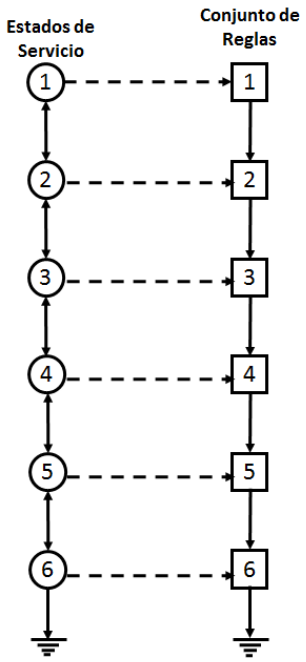


Figura 4.2: Estados de servicio

4.2. Formulación del Problema de Mitigación de Anomalías

Con base en el concepto de estados de servicio introducido en la sesión anterior, el módulo de mitigación de anomalías para un *Firewall* de red se basa en la reasignación de las reglas que deben ser procesadas en cada estado de servicio, a fin de minimizar el número de operaciones que debe ejecutar el procesador (utilización de recursos computacionales) en la interrogación de las reglas hasta un evento de coincidencia. Las anomalías en la utilización de los recursos que se estudian en este trabajo, son ocasionadas principalmente por ataques *DoS* que apuntan a las últimas reglas; cabe anotar que un atacante puede descubrir una de las últimas reglas del *Firewall* mediante ataques de prueba [40] [66] [39], para luego ejecutar un ataque de baja tasa apuntando a la regla descubierta.

Una vez identificada la anomalía, la reasignación de las reglas en los estados de servicio ayudaría a bajar considerablemente el tiempo de *CPU* de cada paquete que es interrogado en el conjunto de reglas, y por tanto la utilización de los recursos. La idea es que los primeros estados de servicio procesen aquellas reglas con mayor probabilidad de coincidencia.

4.2.1. Control de flujo

Siguiedo el modelo como sistema de encolamiento para el *Firewall* mencionado anteriormente, nuevos paquetes son admitidos a la cola del *Kernel*, representados por $B(t)$, de acuerdo a la solución del siguiente problema:

$$\text{Minimizar}: \quad B(t)[Q(t) - Q_{max}] \tag{4.1}$$

$$\text{Sujeto a}: \quad 0 \leq B(t) \leq A(t) \tag{4.2}$$

En la función objetivo 4.1 del problema de optimización, Q_{max} representa la capacidad máxima de la cola, estableciendo la cantidad máxima de paquetes que pueden ser admitidos; por su parte, $Q(t)$ representa el tamaño o número de paquetes en la cola para el intervalo de tiempo t. De acuerdo a esto, si $Q_{max} > Q(t)$, la decisión al procedimiento de control de flujo es $B(t) = A(t)$. Tal resultado indica que el algoritmo de control admite todos los paquetes que arriban a la cola dados

por $A(t)$, cuando el número de paquetes contenidos en la cola es menor que su capacidad para el intervalo de tiempo t. Por el contrario si $Q_{max} \leq Q(t)$, ningún paquete es admitido, siendo $B(t) = 0$.

4.2.2. Dinámicas de encolamiento

Sea $Q(t)$ el tamaño de la cola o búfer del *Kernel* en el intervalo de tiempo o ciclo de reloj t, este búfer evoluciona de acuerdo a la siguiente dinámica de encolamiento:

$$Q(t+1) = max[Q(t) - c_m(t), 0] + B(t) \tag{4.3}$$

donde,

$$c_m \in \{0, 1\},\ m = 1, 2, ..., M \tag{4.4}$$

$$B(t) \leq A(t) \tag{4.5}$$

Aquí, c_m es una variable binaria que toma el valor de uno si en el estado de servicio s_m, su regla asociada $r_j, j = 1, 2, ..., M$ estableció coincidencia con el paquete siendo interrogado en el intervalo de tiempo t, cero en otro caso. En el intervalo de tiempo $t+1$, el número de paquetes almacenados en el búfer del *Kernel* estará dado por el valor maximo entre $Q(t) - c_m(t)$ y cero, más el número de paquetes admitidos en ese instante de tiempo, representados por $B(t)$. Si en t la cola está vacía, es decir, $Q(t) = 0$, la función $max[Q(t) - c_m(t), 0] = 0$ evitando que el computo del tamaño de cola adquiera un valor negativo; por el contrario, si $Q(t) > 0$, el valor $max[Q(t) - c_m(t), 0] = Q(t) - c_m(t)$, la diferencia entre el número de paquetes en cola en el tiempo actual, y el indicador de coincidencia del paquete que se encuentra actualmente en servicio; ésto último en virtud de que una vez se da una coincidencia el paquete que se encuentra en servicio sale del sistema (una decisión de permitir su acceso a la red, o de eliminarlo es tomada según la política de seguridad establecida), y de haber paquetes en cola el primero de éstos pasa a servicio.

Además, se define $X_j(t),\ j = 1, 2, ..., M$ como la cola virtual de registro de coincidencia de la regla $r_j \in R$; se dice que son virtuales en el sentido en que son computadas puramente en software,y

evolucionan de acuerdo a la siguiente dinámica de encolamiento:

$$X_j(t+1) = max[X_j(t) - \rho_m(t), 0] + c_m(t) \tag{4.6}$$

La variable $\rho_m(t)$ representa la utilización de la CPU en la ejecución de las tareas correspondientes a la interrogación de las reglas asociadas a los estados de servicio $s_1, ..., s_m$, siendo s_m el estado de servicio de coincidencia, con regla asociada r_j. Ésto es, se trata de la computación de la utilización de la CPU en el procesamiento de las reglas de los estados de servicio hasta el evento de coincidencia.

4.2.3. Formulación del problema de reasignación de reglas

Este problema de asignación se basa en establecer la regla que debe ser interrogada en cada uno de los estados de servicio,tal que solucione de manera óptima el problema de optimización que se presenta a continuación:

$$\text{Maximizar:} \quad \sum_{n'} \sum_{n} \gamma_{n'n}(t) W_{n'n}(t) \tag{4.7}$$

Sujeto a:

$$\gamma_{n'n}(t) \in \{0,1\} \ \forall \ n', n \ \in \{1,2,...,M\} \tag{4.8}$$

$$0 \leq \sum_{n'=1}^{M} \gamma_{n'n}(t) \leq 1 \ \forall \ n \ \in \{1,2,...,M\} \tag{4.9}$$

$$0 \leq \sum_{n=1}^{M} \gamma_{n'n}(t) \leq 1 \ \forall \ n' \ \in \{1,2,...,M\} \tag{4.10}$$

Constituye un problema de optimización lineal entero, donde la variable de decisión $\gamma_{n'n}$ toma el valor de uno si la regla $n, n = 1,2,...,M$ ha sido asignada al estado de servicio $n', n' = 1,2,...,M$, cero en otro caso, como se establece en la restricción 4.8. La función objetivo 4.7 busca maximizar el peso entre un estado de servicio y cada una de las reglas, dado por $W_{n'n} = X_n(t), \forall \ n = 1,2,...,M$. Las restricciones 4.9 y 4.10 garantizan que a un estado de servicio no se asigne más de una regla, y que una regla no tenga correspondencia con más de un estado de servicio, respectivamente.

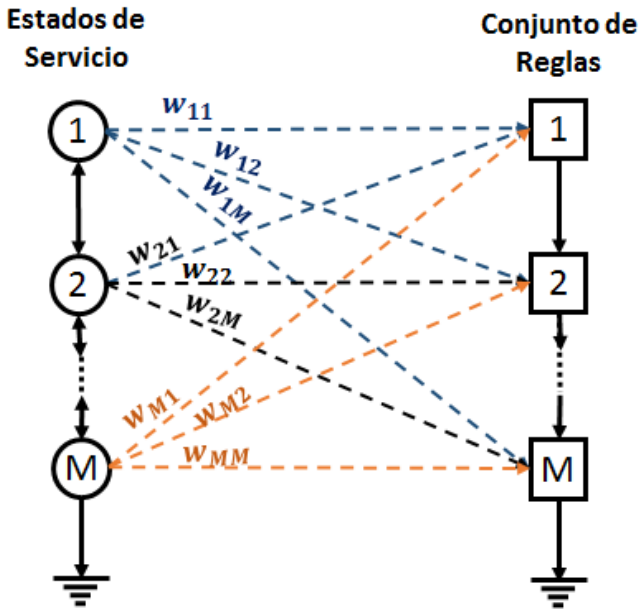


Figura 4.3: Problema de Optimización *Maximum Weight Match* para la asignación de las reglas

Este constituye un problema particular del *Maximum Weight Match* [67][68] sobre un grafo bipartito de M estados de servicio y M reglas como se muestra en la figura 4.3, donde el peso de cada enlace (j', j) en la función objetivo 4.7 está dado por la cola de virtual de registro de coincidencia de regla $X_n(t), n \in 1, 2, ..., M$ que evoluciona de acuerdo a 4.6. De esta forma, la regla con mayor registro de coincidencias tiene mayor prioridad de asignación.

La complejidad computacional que supone este tipo de problemas ha sido ampliamente estudiado [69]. En varios antecedentes se ha demostrado que este es uno de los problemas de optimización combinatorios más fuertes, que puede ser solucionado en tiempo polinomial [67], [70].

4.2.4. Algoritmo *Greedy Maximal Match Scheduling* para la mitigación de anomalías en un *Firewall* de red

Como se mencionó arriba, el problema de asignación de reglas consiste en solucionar el problema de *Maximum Weight Match* sobre un grafo bipartito entre M estados de servicio y M reglas (figura 4.3). No obstante y debido a la complejidad computacional de los algoritmos para la solución de

este problema (tiempo polinomial), se propone la siguiente alternativa de implementación simple que permita obtener una buena solución en términos de tiempo de computo eficientes.

Dicha alternativa está basada en las nociones de *Greedy Maximal Match Scheduling* (*GMMS*) para la asignación del orden de procesamiento de las reglas en cada uno de los estados de servicio, que procure una buena solución en términos de optimalidad y eficiente en términos de complejidad computacional.

El peso del enlace entre el estado de servicio n' y la regla n, según el grafo bipartito definido, está dado por $X_n(t)$: la cola virtual de coincidencia de cada una de las reglas. Con este valor de peso entre los enlaces se busca dar prioridad en la asignación a las reglas que presenten mayor registro de coincidencia, para los primeros estados de servicio. Habiendo definido el conjunto de enlaces $(n', n), n', n = 1, 2, ..., M$ del *maximal match*, se selecciona el enlace $(i, j), i, j = 1, 2, ..., M$ que posee el mayor peso entre los demás enlaces posibles y se marca como activo, indicando que al estado de servicio s_i le ha sido asignada la regla r_j. Consecuentemente, el enlace que posee el segundo mayor peso es marcado como activo bajo la restricción de que no ocasione conflictos con enlaces que hayan sido marcados como activos previamente. Se continúa de esta forma hasta que ningún otro enlace pueda marcarse como activo, es decir, cuando todas las reglas hayan sido asignadas a los estados de servicio. En tal instancia se logra la situación deseada en la que el objetivo de maximización es alcanzado. El algoritmo en 1 describe la estructura del *GMMS*; su implementación es relativamente más fácil que el *Maximum Weight Match*, requiriendo muchos menos recursos computacionales que el primero [71].

En la figura 4.4 se muestra un ejemplo de asignación bajo el algoritmo *GMMS*, que busca solucionar el problema de optimización de 4.7. Se realiza una ilustración del resultado del procedimiento de asignación de recursos con base en los valores de la función de peso del algoritmo *GMMS* para cada par estado de servicio-regla (flechas de color verde para evidenciar la asignación de una regla al determinado estado de servicio. En la parte derecha del gráfico se muestra el conjunto de reglas disponibles para la asignación a los estados de servicio y las dinámicas de encolamiento virtuales de coincidencia de cada regla.

Algorithm 1 *Greedy Maximal Match Scheduling: GMMS(t)*

Step_0 *Input* : $W_{n'n}(t)$;
Initialization : $\gamma_{n'n}(t) := 0 \ \forall n', n$; $\gamma_n := 0 \ \forall n$;

Step_1 **while** $(\exists n \mid \gamma_n == 0)$
$(n'_0, n_0) := \underset{n',n}{argmax}\{W_{n'n}(t)\}$;
$\gamma_{n_0} := 1$;
if $(W_{n'_0 n_0}(t) > 0)$ **then**
$\gamma_{n'_0 n_0}(t) := 1$;
$W_{kn_0}(t) := 0 \ \forall k \in \{1, \ldots, M\}$;
$W_{n'_0 q}(t) := 0 \ \forall q \in \{1, \ldots, M\}$;
endif
endwhile

Step_2 *Output* : $\gamma_{nm}(t)$;

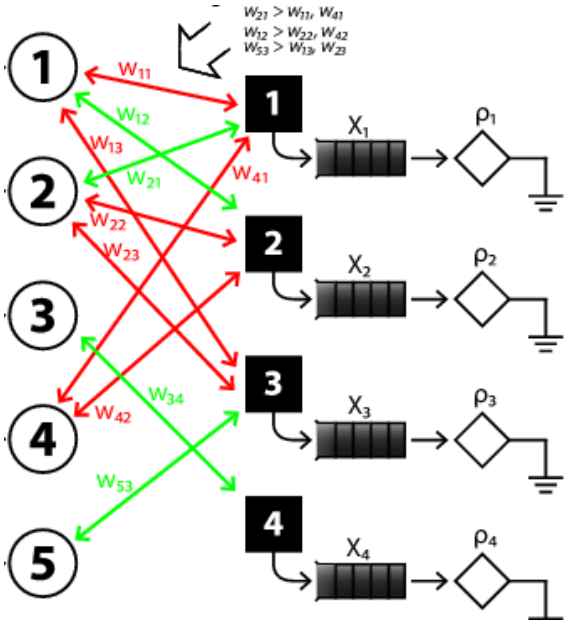


Figura 4.4: Greedy Maximal Match

4.2.5. Resultados de la estrategia de mitigación de anomalías ante ataques *DoS*

En el capítulo 3 se introdujo una metodología para la detección de anomalías en la utilización de los recursos computacionales de un *Firewall* de red, en su mayoría ocasionados por ataques *DoS* apuntando a las últimas reglas del conjunto que definen el mecanismo de seguridad de este importante servicio.

La detección de anomalías se basa en el monitoreo de la métrica utilización de *CPU*; si supera un valor umbral definido, el *Kernel* lanza una alarma de ocurrencia de anomalía que permite la ejecución del algoritmo *GMMS*. De esta forma, todos los estados de servicio de S tendrán asignada una nueva regla que deberá procesar; cabe recordar que s_1, el primer elemento del conjunto S, es el primer estado de servicio que debe ejecutar el *Firewall* en el procesamiento de las reglas, justo después del procesamiento de tareas de red realizado por el *Kernel*. Se espera que con este algoritmo los primeros estados de servicio tengan asignadas las reglas con mayor registro de coincidencia. Así, un ataque que apuntaba a una de las reglas del final del conjunto, y que debía procesar todos los estados de servicio de forma secuencial hasta el evento de coincidencia, esta vez será asignadas a los primeros estados de servicio, bajando considerablemente el tiempo de *CPU* de procesamiento en la interrogación de cada paquete, y en consecuencia, la utilización de la *CPU*.

A fin de analizar el comportamiento del componente de mitigación de anomalías, se estabeció un marco experimental utilizando el simulador descrito en A.1. Se configuró un servidor de 2.4 Ghz de frecuencia de reloj, con capacidad de 512 paquetes en el búfer del *DMARing*; la unidad de procesamiento para las operaciones de red del *Kernel*, fue configurada para atender paquetes con un tiempo de servicio medio de 2.65 μseg; se configuraron 10000 reglas dentro del mecanismo de seguridad, cada una de ellas con un tiempo medio de servicio de 0.05 μseg en la interrogación de los paquetes que ingresan al sistema.

En un primer experimento el *Firewall* de red fue sometido a un flujo de tráfico normal durante todo el tiempo de de observación, con paquetes configurados para establecer coincidencia con reglas ubicadas el el intervalo $[1, 1000]$, y una tasa de 1000 *pps*. En el segundo 80 se ejecuta un ataque

DoS apuntando a la regla 9000, a una tasa de 4000 *pps*.

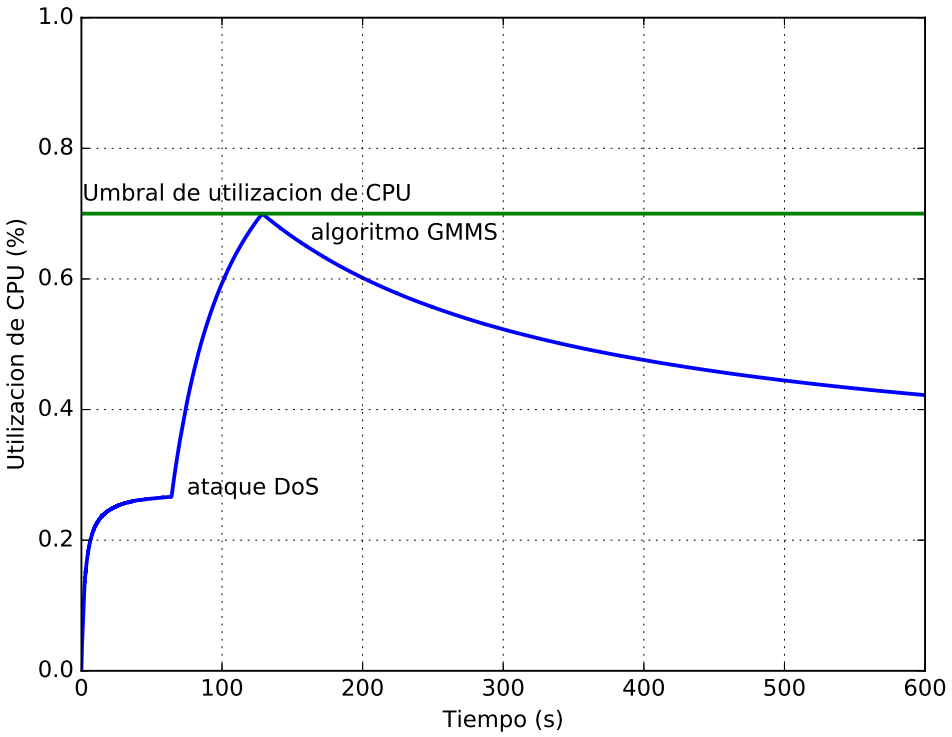


Figura 4.5: Mitigación de anomalías a través del algoritmo *GMMS*

Como se evidencia en la figura 4.5, una vez comienza el ataque la utilización de la *CPU* aumenta drásticamente; el módulo de detección de anomalías, através del monitoreo de la utilización de *CPU*, ejecuta el algoritmo *GMMS* cuando la utilización de la *CPU* alcanza el umbral definido en el 70 %. Una vez este algoritmo asigna las reglas a los estados de servicio, la utilización de la *CPU* retorna a valores de comportamiento normal estabilizandose en un valor del 40 %, mitigando claramente la anomalía.

Lo anterior obliga a los atacantes a ejecutar un nuevo ataque de comprobación a fin de verificar una de las reglas ejecutadas en los últimos estados de servicio, y lanzar ataques posteriores con misma intención de saturar los recursos computacionales del *Firewall* de red.

La figura 4.6 muestra el comportamiento del *Firewall* cuando es sometido a dos ataques *DoS*, ocurriendo en dos instantes de tiempo diferentes dentro del intervalo de observación. Ejecutados a los 30 y 200 segundos, los ataques fueron configurados para apuntar a las reglas 8000 y 10000 respectivamente; ambos a una tasa de 4000 *pps*. Un flujo de tráfico normal de tasa 10000 *pps*, con incidencia durante todo el intervalo de observación, es configurado para que sus paquetes establezcan coincidencia con reglas en el intervalo $[1, 1000]$; antes de entrar a operar el algoritmo de mitigación de anomalías, los estados de servicio que ejecutan estas reglas corresponden a los primeros mil $(s_1, s_2, ..., s_{1000})$.

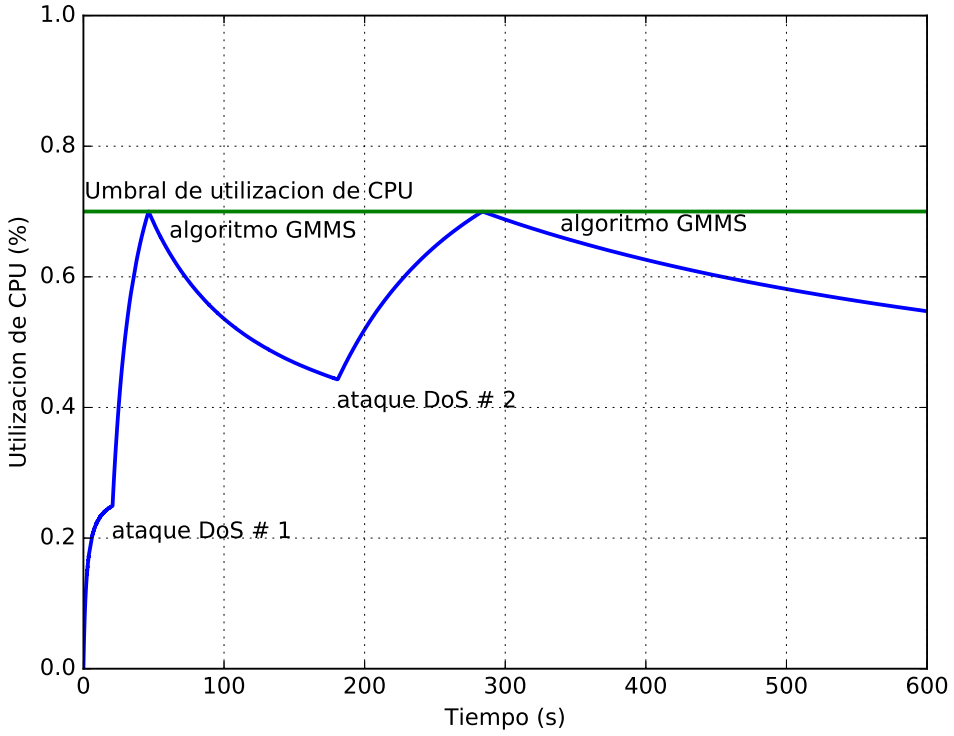


Figura 4.6: Mitigación de anomalías a través del algoritmo textitGMMS

El primer ataque logra "disparar" la utilización de la *CPU*en un tiempo relativamente corto; en el capítulo 2 se mostró como un ataque apuntando a reglas del final del conjunto (5000 a 10000),

alcanza niveles de saturación de 100 % de utilización de la *CPU*, aún para tasas paquetes relativamente bajas. Al alcanzar el valor umbral el *Kernel* ejecuta el algoritmo *GMMS*; la asignación de las reglas a los estados de servicio logra estabilizar el consumo de los recursos, bajando la métrica de utilización de *CPU* a un valor alrededor del 40 %.

El segundo ataque es lanzado cercano a los 200 segundos, saturando los recursos del *Firewall*. Siguiendo la metodología propuesta en este trabajo, la componente de detección de anomalías monitorea en todo momento el consumo de los recursos computacionales; al alcanzar nuevamente el valor umbral, se ejecuta el algoritmo y la utilización de la *CPU* retorna a valores normales.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

El *Firewall* constituye un elemento crucial en el esquema de seguridad de una red de datos. Configura la primera línea de defensa contra los flujos de tráfico que pretenden ingresar al sistema, en muchos casos tráfico malicioso con el objetivo de ocasionar un impacto que puede ser debastador.

Es necesario entonces adoptar alternativas que proveean la capacidad a este sistema de seguir operando, incluso, bajo presencia de eventos que ponen en riesgo su funcionalidad; metodologías de resiliencia que involucren etapas de detección, mitigación e identificación de desafíos.

Los ataques *DoS* contra los *Firewall* de red, son un tipo de flujo de tráfico malicioso que busca generar anomalías en el consumo de los recursos computacionales del servidor que alberga a este importante servicio, hasta niveles que ocasionen un colapso del sistema. Luego de un modelamiento y estudio del rendimiento a través de Cadenas de Markov de tiempo continuo, y asumiendo tiempos entre llegada de paquetes y de atención distribuidos exponencialmente para efectos de la transición de estados, fue posible analizar el rendimiento de este sistema en diferentes escenarios. Puede concluirse que en condiciones normales el *Firewall* puede alcanzar hasta un 40 % en la utilización de la *CPU*, lo que puede caracterizarse como comportamiento normal. Así, una anomalía en la utilización de los recursos se presenta cuando esta métrica supera un valor del 70 % 80 %, debido

quizá a un ataque *DoS*.

Se propuso una metodología para la detección de anomalías que consiste en el monitoreo de la utilización de la *CPU*. Se ha establecido un umbral de utilización de la *CPU*; si la etapa de monitoreo arroja un valor superior a este umbral, se dispara una alarma que informa acerca de la ocurrencia de una anomalía; ésto abre la posibilidad de implemetar medidas de remediación que eviten la salida de operación del *Firewall*.

La definición de los estados de servicio de las reglas, como estructura paralelas que indican la etapa de servicio que debe ejecutarse, permite definir la regla que se procesará en cada uno de los estados. Esto abre la posibilidad de implementar algoritmos de decisión en procura de reducir los tiempos de procesamiento por paquete.

Definidos los estados de servicio y mediante el modelamiento del *Firewall* como un sistema de encolamiento con múltiples etapas de servicio, es posible establecer una estrategia para la remediación de anomalías, basada en asignar las reglas que presenten el mayor registro de coincidencia a los primeros estados de servicio, a fin de minimizar el tiempo de procesamiento de cada paquete antes de abandonar el sistema. De esta forma se logra "suavizar" la utilización de la *CPU*, llevando el sistema a un estado de consumo de los recursos a niveles normales.

La estrategia de remediación o mitigación de anomalías que se plantea en este proyecto, se basa en la formulación de un problema de optimización para la asignación de las reglas a los estados de servicio. Se logró plantear un problema de optimización lineal entero sobre un grafo bipartito entre los estados de servicio y el conjunto de reglas; este problema es una particularización del problema *Maximum Weight Match*.

La utilización del algoritmo *Greedy Maximal Match Scheduling* como alternativa de solución en los procesos de asignación del orden de verificación de las reglas del *Firewall*, exhibe resultados satisfactorios, más allá de que sea una solución aproximada al esquema de asignación que soluciona óptimamente el problema *Maximum Weight Match*.

La estrategia de remediación de anomalías en la utilización de la *CPU* del *Firewall*, logra reducir esta métrica hasta valores cercanos a los 50 %-60 %, en el escenario propuesto, en el cual se ejecutaron ataques *DoS* apuntando a las últimas reglas del mecanismo de seguridad.

Las etapas de detección temprana y de mitigación o remediación de las anomalías en la utilización de los recursos computacionales del *Firewall*, logran proveer la suficiente resiliencia al *Firewall* de red para mantener su operación incluso ante situaciones adversas que amenacen su operabilidad, como los ataques *DoS*.

5.2. Trabajo futuro

Desde el marco conceptual de la resiliencia en los dispositivos de red, es necesario incluir un modulo para la identificación de las anomalías ocasionadas en la utilización de los recursos del *Firewall*, ocasionadas probablemente por ataques *DoS*. Una vez detectada y mitigada dicha anomalía, es posible hacer una inspección minuciosa a los paquetes que vayan dirigidos a la regla que presenta el mayor registro de coincidencia; si el origen del flujo de tráfico es desconocido, puede tomarse medidas que rechacen futuros flujos de tráfico provenientes del mimos origen.

La metodología desarrollada en el marco de este proyecto para proveer resiliencia a un dispositivo como el *Firewall*, puede ser adoptada y extendida a otras etapas de la red, con miras a conseguir la resiliencia global de la red.

En una etapa posterior de este proyecto se busca instrumentar el código del *Firewall Linux Netfilter* con la metodología desarrollada de detección y mitigación de anomalías. Dado el modelamiento realizado a través de los estados de servicio, es posible incluir listas doblemente enlazadas donde cada elemento contenga un apuntador a la regla que debe procesarse en cada estado de servicio, luego de ejecutar el algoritmo de asignación de reglas. Para esta implementación es necesario modificar y compilar el *Kernel* de Linux para verificar la viabilidad de esta propuesta sobre un sistema real.

Este trabajo ha sido desarrollado para la arquitectura del *Firewall Linux Netfilter*, basado en

Listas de Control de Acceso (ACL, acrónimo en inglés), conocidos como "*Stateless-Firewall*". Una futura extensión de este trabajo considera el modelamiento e implementación de los módulos que se desarrollaron, para un *Firewall* basado en la filosofía de inspección de tráfico "*Stateful-Firewall*", el cual usa información del estado de las conexiones en los flujos de tráfico para establecer decisiones dinámicas de control [72].

Bibliografía

[1] G. Rosenbaum and S. Jha, "Resilience provisioning in provider-based overlay networks," in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, Nov 2005, pp. 6 pp.–432.

[2] A. Schaeffer-Filho, P. Smith, A. Mauthe, D. Hutchison, Y. Yu, and M. Fry, "A framework for the design and evaluation of network resilience management," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 401–408.

[3] J. Stewart, *Network Security, Firewalls and VPNs*. Jones & Bartlett Learning, LLC, 2013. [Online]. Available: https://books.google.com.co/books?id=qZgtAAAAQBAJ

[4] S. Suehring, *Linux Firewalls: Enhancing Security with nftables and Beyond*. Pearson Education, 2015.

[5] E. W. Fulp, "Chapter 6 - firewalls," in *Managing Information Security (Second Edition)*, second edition ed., J. R. Vacca, Ed. Boston: Syngress, 2014, pp. 143 – 175.

[6] K. Salah, K. Sattar, M. Sqalli, and E. Al-Shaer, "A potential low-rate dos attack against network firewalls," *Security and Communication Networks*, vol. 4, no. 2, pp. 136–146, 2011.

[7] Z. Trabelsi, L. Zhang, and S. Zeidan, "Dynamic rule and rule-field optimisation for improving firewall performance and security," *IET Information Security*, vol. 8, no. 4, pp. 250–257, July 2014.

[8] M. Y. Su and S. C. Yeh, "An online response system for anomaly traffic by incremental mining with genetic optimization," *Journal of Communications and Networks*, vol. 12, no. 4, pp. 375–381, Aug 2010.

[9] T. Katic and P. Pale, "Optimization of firewall rules," in *2007 29th International Conference on Information Technology Interfaces*, June 2007, pp. 685–690.

[10] M. Garcia, N. Neves, and A. Bessani, "Sieveq: A layered bft protection system for critical services," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[11] T. Cinkler, P. Demeester, and A. Jajszczyk, "Resilience in communication networks," *Communications Magazine, IEEE*, vol. 40, no. 1, pp. 30 – 32, Jan. 2002.

[12] J. P. Sterbenz, D. Hutchison, E. K. Atetinkaya, A. Jabbar, J. Rohrer, M. Schapller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245 – 1265, Jun. 2010.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[14] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Conditional anomaly detection," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 5, pp. 631–645, May 2007.

[15] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining: Pearson New International Edition.* Pearson Education Limited, 2013.

[16] C. De Stefano, C. Sansone, and M. Vento, "To reject or not to reject: that is the question-an answer in case of neural classifiers," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 30, no. 1, pp. 84–94, Feb 2000.

[17] D. Barbara, N. Wu, and S. Jajodia, "Detecting Novel Network Intrusions using Bayes estimators," in *SIAM International Conference on Data Mining*, 2001.

[18] G. Ratsch, S. Mika, B. Scholkopf, and K. Muller, "Constructing boosting algorithms from svms: an application to one-class classification," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 9, pp. 1184–1199, Sep 2002.

[19] A. Liu, *Firewall Design and Analysis*, ser. Computer and network security. World Scientific Publishing Company, 2011.

[20] S. Acharya, J. Wang, Z. Ge, T. Znati, and A. Greenberg, "Simulation study of firewalls to aid improved performance," in *Simulation Symposium, 2006. 39th Annual*, April 2006, pp. 8 pp.–.

[21] "Chapter 9 - implementing a firewall with ipchains and iptables," in *Hack Proofing Linux*, ser. The Only Way to Stop a Hacker Is to Think Like One, J. Stanger and P. T. Lane, Eds. Burlington: Syngress, 2001, pp. 445 – 506.

[22] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "Fireman: a toolkit for firewall modeling and analysis," in *Security and Privacy, 2006 IEEE Symposium on*, May 2006, pp. 15 pp.–213.

[23] M. Rash, *Linux Firewalls*, 1st ed. San Francisco, CA, USA: No Starch Press, 2007.

[24] S. Seth and M. Venkatesulu, *Transmission and Reception of Packets*. Wiley-IEEE Press, 2008, pp. 697–722.

[25] D. Bovet and M. Cesati, *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[26] L. fei Xuan and P. fei Wu, "The optimization and implementation of iptables rules set on linux," in *Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on*, April 2015, pp. 988–991.

[27] Q.-X. Wu, "The research and application of firewall based on netfilter," *Physics Procedia*, vol. 25, pp. 1231 – 1235, 2012, international Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao.

[28] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1996.

[29] S. Prowell, R. Kraus, and M. Borkin, "Denial of service," in *Seven Deadliest Network Attacks*, S. Prowell, R. Kraus, and M. Borkin, Eds. Boston: Syngress, 2010, pp. 1 – 21.

[30] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[31] J. L. Harrington, "Chapter 7 - denial of service attacks," in *Network Security*, ser. The Morgan Kaufmann Series in Networking, J. L. Harrington, Ed. San Francisco: Morgan Kaufmann, 2005, pp. 161 – 179.

[32] C. Douligeris and A. Mitrokotsa, "{DDoS} attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643 – 666, 2004.

[33] H. Nemati, *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*, ser. Contemporary research in information science and technology. Information Science Reference, 2007.

[34] B. Harris and R. Hunt, "Tcp/ip security threats and attack methods," *Computer Communications*, vol. 22, no. 10, pp. 885 – 897, 1999.

[35] K. Buzzard, "Computer security — what should you spend your money on?" *Computers and Security*, vol. 18, no. 4, pp. 322 – 334, 1999.

[36] N. Hoque, M. H. Bhuyan, R. Baishya, D. Bhattacharyya, and J. Kalita, "Network attacks: Taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol. 40, pp. 307 – 324, 2014.

[37] C. Douligeris and D. Serpanos, *DenialofService Attacks*. Wiley-IEEE Press, 2007, pp. 117–134.

[38] E. Al-Shaer, A. El-Atawy, and T. Samak, "Automated pseudo-live testing of firewall configuration enforcement," *Selected Areas in Communications, IEEE Journal on*, vol. 27, no. 3, pp. 302–314, April 2009.

[39] K. Salah, K. Sattar, M. Sqalli, and E. Al-Shaer, "A probing technique for discovering last-matching rules of a network firewall," in *Innovations in Information Technology, 2008. IIT 2008. International Conference on*, Dec 2008, pp. 578–582.

[40] T. Samak, A. El-Atawy, and E. Al-Shaer, "Firecracker: A framework for inferring firewall policies using smart probing," in *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, Oct 2007, pp. 294–303.

[41] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 1, pp. 12–21, March 2012.

[42] S. Yu, "Ddos attack detection," in *Distributed Denial of Service Attack and Defense*, ser. SpringerBriefs in Computer Science. Springer New York, 2014, pp. 31–53.

[43] P. Du and S. Abe, "Detecting dos attacks using packet size distribution," in *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, Dec 2007, pp. 93–96.

[44] K. Lu, D. Wu, J. Fan, S. Todorovic, and A. Nucci, "Robust and efficient detection of ddos attacks for large-scale internet," *Comput. Netw.*, vol. 51, no. 18, pp. 5036–5056, Dec. 2007.

[45] L. Li and G. Lee, "Ddos attack detection and wavelets," in *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, Oct 2003, pp. 421–427.

[46] S. Ma and C. Ji, "Modeling heterogeneous network traffic in wavelet domain," *Networking, IEEE/ACM Transactions on*, vol. 9, no. 5, pp. 634–649, Oct 2001.

[47] S. Jin and D. Yeung, "A covariance analysis model for ddos attack detection," in *Communications, 2004 IEEE International Conference on*, vol. 4, June 2004, pp. 1882–1886 Vol.4.

[48] H. Hamed, A. El-Atawy, and E. Al-Shaer, "Adaptive statistical optimization techniques for firewall packet filtering," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.

[49] E.-S. El-Alfy and S. Selim, "On optimal firewall rule ordering," in *Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on*, May 2007, pp. 819–824.

[50] A. Tapdiya and E. Fulp, "Towards optimal firewall rule ordering utilizing directed acyclical graphs," in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, Aug 2009, pp. 1–6.

[51] G. Misherghi, L. Yuan, Z. Su, C.-N. Chuah, and H. Chen, "A general framework for benchmarking firewall optimization techniques," *Network and Service Management, IEEE Transactions on*, vol. 5, no. 4, pp. 227–238, December 2008.

[52] Z. Trabelsi, H. Sayed, and S. Zeidan, "Firewall packet matching optimization using network traffic behavior and packet matching statistics," in *Communications and Networking (ComNet), 2012 Third International Conference on*, March 2012, pp. 1–7.

[53] U. Mustafa, M. Masud, Z. Trabelsi, T. Wood, and Z. Al Harthi, "Firewall performance optimization using data mining techniques," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, July 2013, pp. 934–940.

[54] I. Mothersole and M. Reed, "Optimising rule order for a packet filtering firewall," in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, May 2011, pp. 1–6.

[55] null, "Packet flow histograms to improve firewall efficiency," in *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*, Dec 2011, pp. 1–5.

[56] L. Gheorghe, *Designing and Implementing Linux Firewalls with QoS Using Netfilter, Iproute2, NAT and L7-filter*, ser. From technologies to solutions. Packt Publishing, Limited, 2006.

[57] G. Purdy, *Linux iptables Pocket Reference*, ser. Pocket Reference (O'Reilly). O'Reilly Media, 2004.

[58] K. Salah, "Queuing analysis of network firewalls," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, Dec 2010, pp. 1–5.

[59] G. Bianchi, "Performance analysis of the ieee 802.11 distributed coordination function," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 3, pp. 535–547, March 2000.

[60] D. G., *Network Security Attacks and Countermeasures*, ser. Advances in Information Security, Privacy, and Ethics. IGI Global, 2016.

[61] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 99–110.

[62] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 75–86.

[63] C. M. Krishna, *Performance Modeling for Computer Architects*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1995.

[64] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/-Software Interface*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[65] P. G. Clark and A. Agah, "Modeling firewalls for behavior analysis," *Procedia Computer Science*, vol. 62, pp. 159 – 166, 2015, proceedings of the 2015 International Conference on Soft Computing and Software Engineering (SCSE'15).

[66] T. Samak, A. El-Atawy, E. Al-Shaer, and H. Li, "Firewall policy reconstruction by active probing: An attacker's view," in *Secure Network Protocols, 2006. 2nd IEEE Workshop on*, Nov 2006, pp. 20–25.

[67] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 4th ed. Springer Publishing Company, Incorporated, 2007.

[68] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm to achieve 100 % throughput in input-queued switches," in *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 1998, pp. 792–799.

[69] D. Gamarnik, T. Nowicki, and G. Swirszcz, "Maximum weight independent sets and matchings in sparse random graphs. exact results using the local weak convergence method," *Random Structures & Algorithms*, vol. 28, no. 1, pp. 76–106, 2006.

[70] S. Das and K. Kapoor, "A tight lower bound for the weights of maximum weight matching in bipartite graphs," *CoRR*, vol. abs/1605.00406, 2016.

[71] X. Lin and N. Shroff, "The impact of imperfect scheduling on cross-layer rate control in wireless networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, March 2005, pp. 1804–1814 vol. 3.

[72] M. G. Gouda and A. X. Liu, "A model of stateful firewalls and its properties," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 128–137.

[73] M. Rash, *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*, ser. No Starch Press Series. No Starch Press, 2007.

[74] P. L'Ecuyer, "Good parameters and implementations for combined multiple recursive random number generators," *Operations Research*, vol. 47, no. 1, p. 159–164, 1999.

[75] M. Umlauft and P. Reichl, "Experiences with the ns-2 network simulator - explicitly setting seeds considered harmful," *Wireless Telecommunications Symposium, 2007. WTS 2007*, pp. 1–5, April 2007.

[76] R. Rosen, *Linux Kernel Networking: Implementation and Theory*, ser. Books for professionals by professionals. Apress, 2014.

[77] F. Al-Haidari, M. Sqalli, K. Salah, and J. Hamodi, "An entropy-based countermeasure against intelligent dos attacks targeting firewalls," in *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, July 2009, pp. 41–44.

Apéndice A

Modulo de Simulacion de un *Firewall* de red para el Simulador de redes ns-2 (Networ Simulator 2)

ns-2 ha sido adoptado como una de las principales herramientas de experimentación en el desarrollo de redes de telecomunicaciones. Se trata de un software de fuente abierta que cuenta con modulos para la simulación de diferentes tipos de sistemas. En este proyecto se ha desarrollado un modulo para la simulación de un *Firewall* de red, libreria que ha sido adherida al simulador de redes ns-2. La implementación sigue la arquitectura de un *Firewall* basado en reglas (ver sección 1.3.0.1, específicamente la del *Linux Netfilter* [73]. En la figura A.1 se ilustra los componentes que conforman el *Firewall* para ns-2, y que se describen a continuación. Cabe resaltar que este modulo emula el comportamiento de un *Firewall* en su configuración original; es posible además, activar la funcionalidad de detección y de mitigación de anomalías independientemente.

La ejecución de las simulaciones son realizadas hasta lograr un 95 % de confiabilidad en los resultados. Este modulo hace uso del generador de números aleatorios de *ns-2*, el cual implementa el

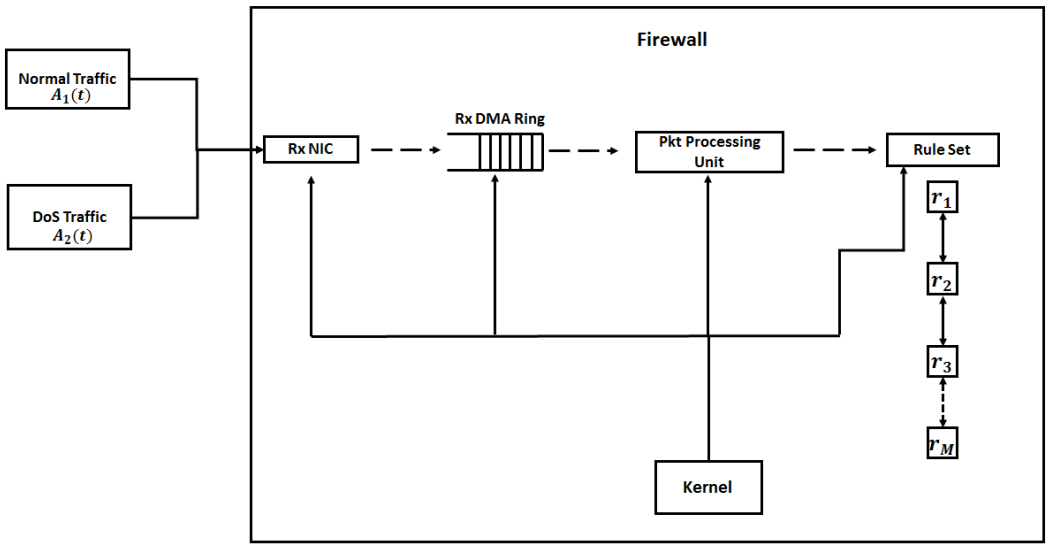


Figura A.1: Estructura del simulador para un *Firewall* de red

generador recursivo *MRG32k3a* propuesto por L'Ecuyer [74]. En [75] los autores muestran como una correcta inicialización de la semilla garantiza resultados no correlacionados.

A.1. Firewall

Es el componente principal, instancia los demás componentes internos que definen la funcionalidad del sistema:

```
$firewall attach-kernel
$firewall attach-rxdmaring
$firewall attach-networking-processing-unit
$firewall attach-ruleset
```

Está compuesto por una tarjeta de red para la recepción de los paquetes (*RxNIC*), un *Kernel* que administra las operaciones que realiza el *Firewall* en todas las etapas, el búfer para el copiado de los paquetes que ingresan al sistema, la unidad de pre-procesamiento para las operaciones de red que se realizan a cada paquete y la lista de reglas del mecanismo de seguridad.

A.2. Kernel

Es el elemento encargado de administrar la tareas procesadas por el *Firewall*[76]; todos los componentes deben referir al *Kernel* como su controlador de dispositivo. En la figura A.1 los enlaces desde el *Kernel* hacia los demás componentes dan cuenta de la relación de dependencia entre estos, en el sentido en el que todas las tareas ejecutadas por cada uno de los componentes es administrada por el *Kernel*:

```
$RxDMARing driver-kernel $kernel
$networkingProcessingUnit driver-kernel $kernel
$ruleSet_ driver-kernel $kernel
```

Su configuración para llevar a cabo una simulación, concibe definir parámetros como la frecuencia de reloj de la *CPU* en GHz.

```
$kernel cpu-clock-rate 2.4
```

También recibe la orden de ejecutar las tareas de monitoreo de la *CPU* para la identificación de anomalías, y para la ejecución del algoritmo de remediación, si es el caso en un escenario de experimentación.

```
$kernel cpu-utilization-monitoring
$kernel cpu-mitigation-algorithm
```

Paralelo al conjunto de reglas que se describe más adelante, el *Kernel* implementa dos listas para la ejecución de las tareas asociadas al procesamiento de las reglas. La primera es la lista de estados de servicio *ServiceStageList*; cada uno de los elementos de esta lista tiene un apuntador a la regla que debe ejecutarse en determinado instante de tiempo; por ejemplo, el primer elemento indica que se trata del primer estado de servicio que debe procesarse, y le corresponde a la regla que tiene asociada ejecutar las tareas de interrogación del paquete. El segundo elemento trata del siguiente estado de servicio que se debe procesar, y tiene asociada la regla a la que le corresponde ejecutar las tareas respectivas; de esa forma se continúa hasta que se presente un evento de coincidencia.

Además, la lista de pesos de regla, *RuleWeightList*, es mantenida en el *Kernel* para efectos de ejecución del algoritmo de remediación de anomalías descrito en la sección 4.2.4.

A.3. Búfer de Recepción de paquetes

Los paquetes admitidos para el procesamiento por parte del *Firewall*, son copiados a la memoria de acceso dinámico del *Kernel RxDMARing*. En este búfer cada paquete espera hasta ser procesado por los diferentes estados de servicio.

En la configuración de este elemento en el simulador, es necesario especificar la capacidad de almacenamiento de paquetes:

```
RxDMARing   buffer-capacity 512
```

La dinámica de encolamiento de este búfer depende de los flujos que arriben al sistema, y del comportamiento de los estados de servicio, unidades que se describen más adelante.

A.4. Unidad de procesamiento de red

Esta unidad es la encargada del preprocesamiento de los paquetes en lo concerniente a la ejecución de las tareas de red.

A nivel de simulación, se asume que el tiempo de servicio de esta unidad es independientemente e identicamente distribuído de acuerdo a una distribución exponencial con parámetro μ, que computa el tiempo que tomarían las tareas ejecutadas en este estado de servicio. El componente recibe dicho parámetro μ como como el tiempo medio de servicio en segundos:

```
NetworkingProcessingUnit mean-service-time 0.00000265
```

A.5. Conjunto de reglas

Este componente contiene las reglas que conforman el mecanismo de seguridad del *Firewall*. Al instanciar este componente, se crean automáticamente las instancias de cada una de las reglas del conjunto, y que serán procesadas, en principio, de manera secuencial según el orden en la lista; si se incluye el modulo de detección y remediación de anomalías, las reglas serán interrogadas de acuerdo al orden establecido por el algoritmo de remediación.

Los parámetros que recibe este componente son el número de reglas que tendra el mecanismo, y la tasa de atención de servicio de las reglas en segundos. Similar a lo establecido para la unidad de procesamiento de red, las tareas ejecutadas para cada uno de los estados de servicio correspondientes a las reglas, son computadas mediante un tiempo de servicio que se asume independientemente e identicamente distribuído de acuerdo a una distribución exponencial con parámetro r.

```
RuleSet setrules 10000
RuleSet rule-mean-service-time 0.00000005
```

A.6. Fuentes de tráfico

El *Firewall* de red puede estar sometido a flujos de tráfico normal o malicioso. Se asume que estan dadas como un proceso de llegada de *Poisson* con tasa λ_i, con $i = 1, 2, ..$ el indice de la fuente de tráfico. En [41], [77], [6] se realiza experimentación en simulación asumiendo fuentes de de tráfico de *Poisson*; aunque este tipo de tráfico no abarca las características en ráfaga del tráfico de internet, los resultados obtenidos en esos trabajos comparan los resultados de simulación vs resultados en equipos reales, concluyendo de acuerdo a la similitud en los resultados obtenidos acerca de la viabilidad en la utilización como fuentes de tráfico.

Para el caso de flujos de tráfico normal, en la configuración del componente es necesario asignar tanto la tasa de llegada de paquetes, así como el intervalo de reglas entre las cuales cada paquete establecería coincidencia:

```
NormalTrafficFlow arrival-rate 10000
NormalTrafficFlow rule-matching 1 1000
```

Las fuentes de tráfico malicioso o ataques *DoS* deben configurar, por su parte, la tasa de llegada de paquetes y la regla a la que va dirigida el ataque:

```
DoSTrafficFlow arrival-rate 4000
DoSTrafficFlow rule-target 9000
```

Se ha definido un valor de U_{max} = 70 %; el ataque *DoS* es ejecutado a los 30 segundos; alrededor de los 90 segundos la utilización de la *CPU* alcanza el valor umbral, en tanto el *Kernel* es notificado del evento, permitiendo la ejecución de una posible estrategia de remediación que permita soportar este evento infortunado sin la salida de operación del *Firewall*.

Cabe resaltar que este tipo de ataques *DoS* no son la única causa que puede generar anomalías en la utilización de la *CPU* del dispositivo; altas tasas de paquetes de tráfico normal o carga excesiva del procesador debido a otras tareas que se estén ejecutando, también pueden generarlas eventualmente. Así, este módulo está basado en determinar cuando la utilización de los recursos del *Firewall* alcanza niveles de saturación peligrosos para su operación, como indicador de la ocurrencia de la anomalía independiente de la causa.

Como se evidenció en el análisis de rendimiento del capítulo 2, un ataque de este tipo apuntando a una de las últimas reglas del conjunto logran llevar el consumo de recursos a niveles dramáticamente altos en poco tiempo, aún con una tasa de paquetes relativamente baja.