UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

ANÁLISIS Y MODELOS DE DATOS DE REDES PARA SEGURIDAD INFORMÁTICA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

JOAQUÍN GONZALO CHÁVEZ BARBASTE

PROFESOR GUÍA:
ALBERTO CASTRO ROJAS

MIEMBROS DE LA COMISIÓN:
CÉSAR AZURDIA MEZA
CLAUDIO ESTÉVEZ MONTERO

SANTIAGO DE CHILE
2016

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: JOAQUÍN GONZALO CHÁVEZ BARBASTE
FECHA: ENERO 2016
PROF. GUÍA: ALBERTO CASTRO ROJAS

ANÁLISIS Y MODELOS DE DATOS DE REDES PARA SEGURIDAD INFORMÁTICA

Hoy en día son cientos los servicios que se ofrecen de manera virtual a través de Internet, muchas veces exigiendo la transferencia y almacenamiento de datos prioritarios o sensibles a través de las redes de comunicaciones. Esto obliga a que los factores de seguridad, estabilidad y confiabilidad de las plataformas sea un elemento clave a considerar en el desarrollo de las aplicaciones y, por lo tanto, la utilización de herramientas dedicadas a la prevención y detección de fallas de seguridad toma un rol crítico.

Considerando lo anterior, este trabajo propone diseñar e implementar un ambiente de simulación de redes que permita explorar la aplicación de distintas técnicas de inteligencia computacional en tópicos de seguridad informática, enfocándose principalmente en la utilización de modelos autoregresivos integrales de media móvil (conocidos como ARIMA por su sigla en inglés) y la divergencia de Kullback-Leibler para la detección de ataques de denegación de servicio a un servidor web. Para esto se desarrolla una arquitectura representativa de un proveedor de servicios de Internet utilizando como plataforma el programa de simulación de redes conocido como GNS3 (*Graphical Network Simulator 3*), en donde se tienen tres *routers*, un simulador de clientes, una máquina atacante, un servidor web y una máquina dedicada a la recolección y análisis de datos, todos funcionando como máquinas virtuales dentro de un mismo computador principal. Utilizando este ambiente se ejecutan múltiples simulaciones de tráfico web para condiciones normales de operación y bajo ataque de denegación de servicio, obteniéndose series de tiempo de alrededor de doce horas de duración para capturar el comportamiento a nivel IP del tráfico durante períodos de volumen de usuarios con carga baja, media y alta. De esta manera se logra recolectar suficientes datos para poder realizar el análisis estadístico respectivo y la generación de modelos de predicción de tráfico y detección de ataques con la ayuda del lenguaje para análisis estadístico R.

A partir de los resultados obtenidos se verifica la factibilidad de utilizar modelos ARIMA para la predicción del tráfico que fluye a través de los *routers* que conforman las vías troncales de la red y como herramienta complementaria de detección de cambios bruscos en el nivel de tráfico de subida a un servidor web. Además, se obtienen resultados exitosos para la utilización de la divergencia de Kullback-Leibler como mecanismo de detección de ataques de denegación de servicio, en base a los cambios bruscos de tráfico, permitiendo la detección dentro de los primeros cinco minutos de comenzada la falla. Este tipo de herramientas permitirían a los proveedores implementar sistemas inteligentes para la detección temprana de ataques de denegación de servicio dentro de su red, pudiendo aplicar maniobras de mitigación a tiempo y, por lo tanto, fortaleciendo la seguridad del sistema en su totalidad.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

Las redes de comunicaciones son cada día más importantes en el desarrollo de la sociedad, ya que han pasado a formar parte fundamental de los procesos comerciales, industriales, económicos y sociales de ésta. En efecto, hoy en día son cientos los servicios que se ofrecen de manera virtual a través de internet y su proliferación se debe no solo a la optimización de procesos y reducción de costos transaccionales que permiten, si no que en gran medida depende de la estabilidad y confiabilidad que entregan a sus usuarios al momento de utilizarlos. Por otro lado, la gran mayoría de ellos, al ser servicios personalizados, exigen la transferencia o almacenamiento de datos prioritarios o sensibles a través de la red, por lo tanto el factor de seguridad es un elemento clave a considerar en su desarrollo y la utilización de herramientas dedicadas a este propósito toma un rol crítico.

En este contexto es que este trabajo propone llevar los análisis estadísticos ampliamente utilizados en distintas disciplinas como medicina, control automático, procesos industriales, etc., a través de modelos de datos a la seguridad informática en las redes de comunicaciones, con el objetivo de explorar soluciones de este tipo aplicadas a la detección y predicción de eventos de inseguridad.

En particular, lo que se busca es analizar la factibilidad de utilizar modelos autoregresivos integrales de media móvil (ARIMA por su sigla en inglés) para predicción de tráfico, además de la divergencia de Kullback-Leibler como mecanismo de detección de ataques de denegación de servicio (DoS), por lo que es necesario en un principio diseñar e implementar un ambiente de simulación que permita el estudio de distintos escenarios utilizando un computador personal, para luego poder generar y procesar los datos necesarios para el estudio.

1.2. Alcances y Objetivo General

El trabajo desarrollado comprende el diseño e implementación de un ambiente de simulación basado en herramientas de código libre para el estudio de una red simplificada de Internet en cuanto a servicios, aplicaciones, flujos y vulnerabilidades de seguridad. Utilizando estas herramientas se estudia el comportamiento de la red durante condiciones de tráfico web normal y durante ataques informáticos definidos, con el objetivo de generar modelos de predicción y detección que permitan detectar la ocurrencia de un ataque informático mientras éste está en curso o en el corto plazo, desde su comienzo.

1.3. Objetivos Específicos

Los objetivos específicos para el desarrollo de este trabajo son los siguientes:

- Diseñar e implementar una arquitectura de red simplificada y representativa de Internet utilizando GNS3 como software base.
- Implementar generadores de tráfico web utilizando herramientas de software y un modelo simple propuesto para el comportamiento de los usuarios.
- Investigar sobre ataques informáticos para ejecutarlos durante las simulaciones.
- Recolectar datos sobre los flujos de información a través de los *routers* para ser analizados posteriormente.
- Generar modelos SARIMA o ARIMA para predicción de tráfico a través de los *routers*.
- Generar un modelo de detección de ataques basado en la divergencia de Kullback-Leibler.
- Evaluar cualitativamente la factibilidad de usar estas metodologías como alternativa complementaria a los mecanismos tradicionales para la prevención y detección de ataques a un servicio o aplicación web, actuando siempre desde el punto de vista del proveedor de servicios de Internet.

1.4. Estructura del Documento

- **Capítulo 1. Introducción:** En este capítulo se presenta una breve descripción introductoria, además de la motivación, alcances y objetivos del trabajo realizado.
- **Capítulo 2. Antecedentes:** Este capítulo entrega un resumen de la información bibliográfica y los conceptos básicos más importantes para comprender el trabajo desarrollado. Se incluyen tópicos de estadística, inteligencia computacional y seguridad informática.

- **Capítulo 3. Metodología:** En este capítulo se describe en detalle el flujo de trabajo para lograr cada uno de los objetivos específicos del estudio, incluyendo el desarrollo del ambiente de simulación, la generación de datos y el posterior análisis de éstos.
- **Capítulo 4. Resultados y Discusión:** Este capítulo presenta los resultados más importantes obtenidos a partir del trabajo realizado. Se incluyen resúmenes de los datos obtenidos a partir de las simulaciones y detalles de los modelos obtenidos a partir de éstos. Además se presenta el análisis y discusión correspondiente en cada caso.
- **Capítulo 5. Conclusiones:** Finalmente en este capítulo se detallan las conclusiones obtenidas y aprendizajes logrados a partir del trabajo realizado en su totalidad y en particular a partir de los resultados presentados en el capítulo anterior. Además se añaden algunas recomendaciones y sugerencias para trabajos futuros ligados a los tópicos presentados en este documento.

Capítulo 2

Antecedentes

En este capítulo se presenta el contexto y concepto teóricos necesarios para la realización y comprensión del trabajo realizado, incluyendo fundamentos y definiciones sobre seguridad informática en redes de comunicaciones, modelos de predicción, detección y clasificación, y la aplicación de éstos para la prevención y detección de ataques informáticos.

2.1. Modelos de Predicción

Para efectuar la predicción de tráfico a través de la red se utilizará la aproximación de análisis de series de tiempo para generar modelos lineales, en particular modelos SARIMA o ARIMA.

2.1.1. Series de Tiempo

El estudio de series de tiempo ve su aplicación en distintas áreas de interés, como análisis económicos, proyecciones de ventas, planificación de redes, medicina, ingeniería, etc. Además, dependiendo de los objetivos, se pueden encontrar distintas técnicas para el tratamiento de los datos y obtener así modelos que permitan tomar acciones de predicción, detección y control, según su aplicación.

Una serie de tiempo es una secuencia de observaciones (muestras) de una variable particular [6] donde cada observación se registra dentro de un tiempo específico. En el caso de las series de tiempo discretas, estas observaciones son registradas en períodos de tiempo regulares.

Tipos de Series de Tiempo

Las series de tiempo pueden clasificarse según sus características de construcción de la siguiente manera:

1. **Muestreadas:** Series obtenidas a partir del muestreo de un conjunto de medidas continuas que pueden ser registradas de manera constante por un periodo de tiempo. Corresponde a una discretización de series continuas.
2. **Agregadas:** Generadas a partir de la acumulación o agregación de datos concatenados, obteniéndose series que representan una condición global.
3. **Discretas:** Series de registros consecutivos obtenidos con periodos regulares de tiempo entre cada muestra.

Objetivo del Análisis de Series de Tiempo

Los principales objetivos que se persiguen con el análisis de series de tiempo son:

1. **Descripción:** Caracterizar la serie de tiempo utilizando métodos de descripción estadísticos, gráficos, valores representativos (máx., mín., tendencias, ciclos), etc.
2. **Modelamiento:** Generar modelos que representen el comportamiento de los datos. Estos modelos pueden ser "Univariables", es decir, que consideran solo valores del pasado de una variable específica, o "Multivariable" en el caso en que se consideren valores presentes y pasados de la misma u otras variables asociadas al proceso.
3. **Predicción:** Estimar los valores futuros de una serie de tiempo, manteniendo una representación cercana a la realidad, a partir de valores pasados y presentes.
4. **Control:** A partir de la caracterización del comportamiento de los procesos obtenida desde las series de datos, es posible realizar acciones de control. El control a realizar está estrechamente ligado a la predicción y, por supuesto, al tipo de modelo generado.

Métodos de Descripción de Series

Antes de hacer cualquier análisis definitivo sobre las series de datos, es necesario "procesar" la información para remover errores que se puedan encontrar en ésta, como datos faltantes, mediciones incorrectas, valores fuera de escala, etc. Información específica sobre métodos de procesamiento de los datos puede ser encontrada en las referencias [4], [5].

En una serie de tiempo se pueden encontrar dos componentes de variación que la conforman, la "tendencia" y la "variación estacional", como se muestra en la figura 2.1.

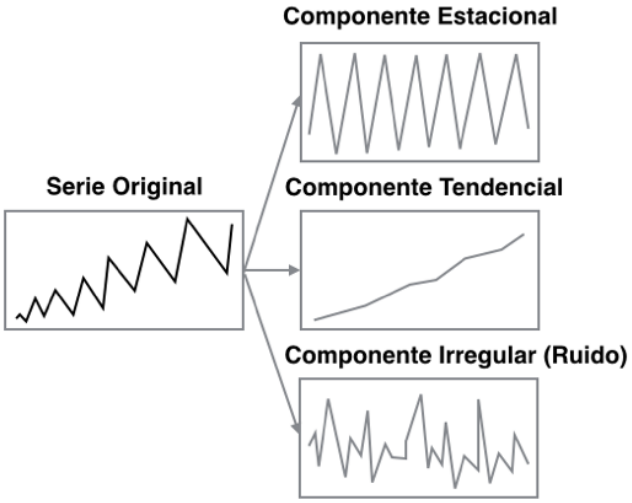

Figura 2.1: Descomposición de series de tiempo.

Como expresión matemática, una serie de tiempo está representada por,

$$X_t = f(S_t, T_t, C_t, I_t) \tag{2.1}$$

donde, cada una de las componentes se define como sigue:

1. **Variación estacional (S_t):** Variación que presenta un patrón de comportamiento similar dentro de un periodo largo de tiempo. Por ejemplo, oscilaciones anuales, mensuales, semanales, etc.

2. **Tendencia (T_t):** Variación que refleja el incremento o reducción constante de los valores de la serie de tiempo dentro de un periodo de muestra o dentro de una variación estacional.

3. **Variación cíclica (C_t):** Incluye variaciones de los datos en periodos de magnitud mayor. Generalmente se asocia a fenómenos cíclicos como procesos económicos de décadas, fenómenos naturales, comportamientos biológicos, etc.

4. **Fluctuaciones (I_t):** Factor que describe las variaciones remanentes después de remover las componentes tendenciales, estacionales y otros valores sistemáticos. Estas fluctuaciones pueden ser completamente aleatorias, en cuyo caso no es posible pronosticarlas.

2.1.2. Modelos Lineales [4]

Proceso Auto Regresivo (AR)

Una serie de tiempo Z_t se dice que es un proceso auto regresivo de orden p ($AR(p)$) si el proceso cumple con la siguiente expresión:

$$Z_t = \sum_{k=1}^{p} \phi_k Z_{t-k} + e_t \tag{2.2}$$

donde ϕ_k corresponde a un factor escalar y e_t es un ruido de media cero y varianza σ_e^2. En otras palabras, el valor de la serie de tiempo en el tiempo t es igual a una combinación lineal de p valores anteriores de la serie, más una componente de ruido.

Proceso de Media Móvil (MA)

Una serie de tiempo Z_t se dice que es un proceso de media móvil de orden q ($MA(q)$) si el proceso cumple con la siguiente expresión:

$$Z_t = \sum_{k=1}^{q} \theta_k e_{t-k} + e_t \tag{2.3}$$

donde θ_k corresponde a un factor escalar y e_t es un ruido de media cero y varianza σ_e^2. Por lo tanto, el valor de la serie en el tiempo t es una combinación lineal de los q valores de ruido anteriores más una componente nueva de ruido e_t.

Proceso Auto Regresivo de Media Móvil (ARMA)

Cuando se combinan los dos modelos presentados anteriormente se obtiene un modelo auto regresivo de media móvil. Si el modelo cuenta con p términos auto regresivos y q términos de media móvil ($ARMA(p,q)$), entonces la expresión que lo describe está dada por:

$$Z_t = \sum_{k=1}^{p} \phi_k Z_{t-k} + \sum_{k=1}^{q} \theta_k e_{t-k} + e_t \tag{2.4}$$

Proceso Auto Regresivo Integral de Media Móvil (ARIMA)

Los modelos presentados anteriormente funcionan para modelos estacionarios, pero este no es el caso para la mayoría de las series de tiempo a estudiar. Para solucionar este problema, se proponen los modelos auto regresivos integrales de media móvil (ARIMA), en donde a través de la diferenciación de la serie de tiempo se obtiene un proceso estacionario.

La expresión que describe un modelo ARIMA está dada por:

$$(1-\sum_{k=1}^{p}\phi_k B^k)(1-B)^d Z_t = \sum_{k=1}^{q}\theta_k e_{t-k} + e_t \tag{2.5}$$

donde B corresponde a un operador de retraso para la serie de tiempo como se describe a continuación:

$$BZ_t = Z_{t-1}$$
$$B^d Z_t = Z_{t-d}$$

Con esto, se agrega como nuevo grado de libertad al modelo el valor de d, que corresponde al nivel de diferenciación con el que se genera el modelo.

Proceso Auto Regresivo Integral de Media Móvil Estacional (SARIMA)

Como última generalización, se puede agregar la condición de estacionalidad de la serie de tiempo en estudio, si es que corresponde, generando entonces un modelo ARIMA Estacional (SARIMA por su sigla en inglés). De esta manera se incluye de manera matemática la estacionalidad de los datos en el modelo, obteniéndose la siguiente expresión que lo describe:

$$(1-\sum_{k=1}^{p}\phi_k B^k)(1-B)^d(1-B^s)^D Z_t = \sum_{k=1}^{q}\theta_k e_{t-k} + e_t \tag{2.6}$$

Además se puede generalizar esta expresión de manera multiplicativa obteniéndose:

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D Z_t = \Theta(B^s)\theta(B)e_t$$

donde:

$$\phi(B) = 1-\sum_{k=1}^{p}\phi_k B^k$$
$$\Phi(B^s) = 1-\sum_{k=1}^{p}\Phi_k B^k s$$
$$\theta(B) = 1+\sum_{k=1}^{p}\theta_k B^k$$
$$\Theta(B^s) = 1+\sum_{k=1}^{p}\Theta_k B^k s$$

Identificación de Sistemas

Para identificar las características del sistema a modelar, y por lo tanto los parámetros que se seleccionarán para el modelo, se sugiere el siguiente procedimiento [4]:

1. Detectar estacionalidad a través de la función de autocorrelación.
2. Detectar estacionariedad por medio de la función de autocorrelación y espectro de frecuencias.
3. Diferenciación estacionaria.
4. Diferenciación estacional.
5. Determinación de órdenes del modelo p y q.
6. Análisis de la forma de la función de autocorrelación

2.2. Modelos de Detección

2.2.1. Test de Hipótesis [22]

En estadística se utiliza el método de inferencia conocido como "Test de Hipótesis" para evaluar, a través de una muestra, una afirmación hecha sobre un conjunto de valores o población. Esta afirmación se manifiesta generalmente como un par de estados, hipótesis nula e hipótesis alternativa, basadas en el valor de un parámetro asociado al estudio de las muestras.

En particular, para este trabajo se utiliza esta herramienta como método de decisión entre dos condiciones de operación (normal o bajo ataque) representadas por distribuciones de probabilidad obtenidas a través de muestras de entrenamiento. Para cada muestra de validación se hace la comparación con las condiciones de operación entrenadas, logrando así la detección de un ataque.

$$H_1 : Q(x) \sim P_1(x) \tag{2.7}$$
$$H_2 : Q(x) \sim P_2(x)$$

2.2.2. Errores del Test de Hipótesis

En la prueba de hipótesis siempre se enfrenta la posibilidad de decidir erróneamente al favorecer una hipótesis por sobre la otra, o al no rechazar apropiadamente la hipótesis falsa. Estas posibilidades de error se conocen como errores de tipo I y errores de tipo II.

Error de Tipo I y Error de Tipo II

- El error de tipo I ocurre si la hipótesis H_2 es rechazada cuando ésta es verdadera. Se denomina usualmente como "error de detección".
- El error de tipo II ocurre cuando la hipótesis alternativa H_2 no es rechazada cuando ésta es falsa. La probabilidad de un error de tipo II se denomina usualmente como "falso positivo".

2.2.3. Divergencia de Kullback-Leibler

En la teoría de la información existe una definición elaborada por Solomon Kullback y Richard Leibler [12] para calcular la información contenida en un experimento para distinguir entre dos distribuciones hipotéticas en un espacio muestral. La definición, presentada originalmente por Wiener, corresponde a una generalización de la entropía de Shannon y tiene que ver con las propiedades de información medida como una prueba de hipótesis estadística.

La entropía de una variable aleatoria es una medida de la incertidumbre que ésta tiene [7], lo que tiene relación con la cantidad de información requerida en promedio para describirla. A su vez, la entropía relativa $D(p||q)$ es una medida de la distancia entre dos distribuciones de probabilidad, es decir, permite medir la ineficiencia de asumir que la distribución de una variable es q cuando la verdadera es p. La entropía relativa o divergencia de Kullback-Leibler entre dos funciones de masa de probabilidad está definida por:

$$D(p||q) = \sum_{x \in X} p(x) \log \left[\frac{p(x)}{q(x)} \right] = E_p \log \left[\frac{p(x)}{q(x)} \right] \tag{2.8}$$

donde $p(x)$ y $q(x)$ corresponden a las funciones de masa de probabilidad a comparar para el evento x.

La divergencia es siempre "no negativa" y es cero si y solo si $p = q$. Es importante destacar que esta medida no es una verdadera distancia entre las distribuciones, pues no es simétrica y tampoco satisface la desigualdad triangular. Sin embargo, es útil a la hora de evaluar la distancia entre distribuciones o entropía relativa.

Con esta medida que evalúa la distancia entre distribuciones es posible realizar la detección de un patrón de tráfico según la distribución medida experimentalmente y los datos de entrenamiento que se tienen para el modelo que describe el estado de operación normal y bajo falla.

2.3. Modelos de Clasificación

Para realizar el estudio de ataques a nivel de aplicación se utilizarán dos métodos de análisis de etiquetas para la clasificación y detección de estados de operación. A continuación se describen de manera general, el método de análisis de componentes principales y el método de análisis de correspondencia, como las dos herramientas básicas a utilizar en el desarrollo.

2.3.1. Análisis de Componentes Principales

El análisis de componentes principales [11] es una metodología que permite reducir la cantidad de variables a considerar dentro de un estudio, en base a la influencia que tiene cada una de ellas con respecto a la variable objetivo. Para esto se consideran las varianzas y la estructura de correlaciones entre las p variables aleatorias del vector x en estudio, de manera de realizar una limpieza y conservar solo las variables, o derivados de éstas, que entreguen la mayor parte de la información necesaria para caracterizar el comportamiento del sistema. Por lo tanto, la idea es encontrar $k \ll p$ combinaciones lineales de las p variables del vector, no correlacionadas entre ellas, que sean responsables de la mayor parte de la variación del vector x.

De esta manera el componente principal i estaría definido por la expresión siguiente:

$$\alpha_i' \mathbf{x} = \alpha_{i1} x_1 + \alpha_{i2} x_2 + \cdots + \alpha_{ip} x_p \tag{2.9}$$

$$= \sum_{j=1}^{p} \alpha_{ij} x_j \tag{2.10}$$

donde los valores α_i corresponden a factores escalares de ponderación para cada uno de los componentes del vector de variables $\mathbf{x}$.

2.3.2. Análisis de Correspondencia

El análisis de correspondencia es una técnica de estudio estadístico descriptivo [3] propuesto por Hirschfeld y desarrollado posteriormente por Jean-Paul Benzécri, que conceptualmente es similar al análisis de componentes principales pero aplica al análisis de etiquetas y categorías. Este método trata las filas y columnas de la matriz de datos de manera equivalente, siendo aplicado frecuentemente en tablas de contingencia, descomponiendo el estadístico chi-cuadrado asociado en factores ortogonales.

2.4. Conceptos de Seguridad Computacional

2.4.1. Seguridad Computacional

El libro "Computer Security Handbook" de la NIST [6] define el término "computer security" como: *"Protección provista a un sistema automatizado de información para lograr los objetivos aplicables de preservar la integridad, disponibilidad y confidencialidad de los recursos del sistema de información (incluyendo hardware, software, firmware, información, datos y telecomunicaciones)."*

Esta definición presenta tres objetivos que conforman la base de la seguridad computacional [19],

1. **Confidencialidad:** Este término cubre dos conceptos relacionados:
   - **Confidencialidad de datos:** Asegura que información privada o confidencial no se vuelva disponible o liberada a individuos no autorizados.
   - **Privacidad:** Asegura que los individuos controlen qué información relacionada con ellos pueda ser recolectada y almacenada, además de por quiénes y para quiénes esa información puede ser liberada.
2. **Integridad:** Este término cubre dos conceptos relacionados:
   - **Integridad de datos:** Asegura que la información y los programas sean modificados de manera específicamente autorizada.
   - **Integridad de sistema:** Asegura que un sistema ejecute su función prevista sin defectos, libre de manipulaciones intencionales o involuntarias no autorizadas.
3. **Disponibilidad:** Asegura que el sistema trabaje sin demoras y el servicio no sea negado a usuarios autorizados.

   Estos tres conceptos forman lo que se conoce como la triada CIA, componentes fundamentales para la seguridad de datos y servicios de información y computación.

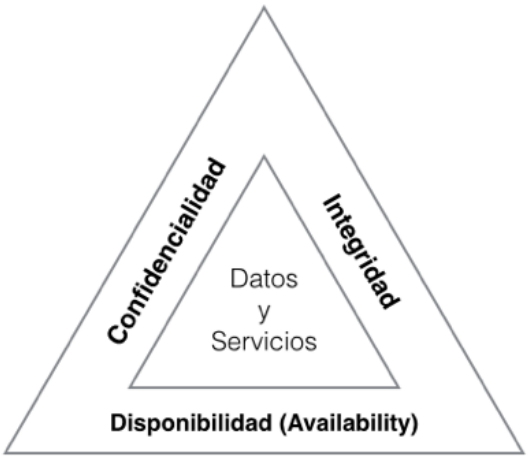

Figura 2.2: Triada de requerimientos de seguridad

Además de los tres objetivos fundamentales, se agregan dos conceptos adicionales que permiten obtener una visión más completa del problema de seguridad informática:

1. **Autenticidad:** La propiedad de ser legítimo y capaz de ser verificado y confiado; confianza en la validez de una transmisión, un mensaje o un mensajero. Esto significa verificar que los usuarios son quiénes dicen que son y que cada entrada que llega al sistema viene de fuentes confiables.

2. **Responsabilidad (*Accountability*):** Es el objetivo de seguridad que genera el requerimiento de trazar las acciones de una identidad de manera única hasta la misma. Esto permite "no repudio", aislamiento de fallas, detección y prevención de intrusiones, etc. Como no existen sistemas totalmente seguros, es necesario poder trazar una brecha de seguridad hasta el responsable para tomar las acciones correspondientes.

2.4.2. Arquitectura OSI de Seguridad

La arquitectura OSI de seguridad estandarizada por la ITU (ITU-T Recommendation X.800) define una aproximación sistemática para los administradores de seguridad y proveedores, que permite organizar la tarea de entregar seguridad informática. Esta arquitectura se enfoca en tres conceptos básicos: ataques, mecanismos y servicios de seguridad.

1. **Ataques de seguridad:** Cualquier acción que comprometa la seguridad de la información perteneciente a una organización. Estos ataques pueden clasificarse en ataques pasivos o activos.

    - **Ataques pasivos:** El objetivo del oponente es obtener información que está siendo transmitida, ya sea a través de la lectura directa de los mensajes o a través de análisis

de patrones de tráfico.

- **Ataques activos:** Ataques que involucran la modificación los datos (o del stream de datos) o la creación de un falso stream de datos. Se subdividen en suplantación (*masquerade*), repetición (*replay*), modificación de mensajes y negación de servicio.

2. **Servicios de seguridad:** Servicio que es provisto por una capa de protocolos de sistemas abiertos de comunicación, que asegura un adecuado nivel de seguridad en los sistemas y transferencias de datos. Algunos servicios de seguridad son los de autenticación, control de acceso, confidencialidad de datos, integridad de datos y de no repudio.

   - **Mecanismos de seguridad específicos:** Pueden ser incorporados en la capa de protocolos para proveer los servicios de seguridad OSI. Incluyen cifrado, firma digital, control de accesos y de ruteo.
   - **Mecanismos de seguridad persuasivos:** No son específicos de un servicio de seguridad OSI o capa de protocolos. Incluyen detección de eventos, etiquetado de seguridad y recolección de datos para auditoría.

3. **Mecanismos de seguridad:** Los mecanismos de seguridad definidos (X.800) se clasifican dependiendo de si se implementan a nivel de un protocolo o servicio específico o no.

2.4.3. Modelo de Seguridad en Redes

En la Figura 2.3 se muestra un modelo general para seguridad en redes de comunicaciones que representa el envío de un mensaje desde un usuario a otro a través de un servicio de Internet. Ambos usuarios son fundamentales en la transacción y deben cooperar para que el intercambio ocurra. Además se establece un canal lógico de transmisión de información por medio de protocolos de comunicación (por ejemplo TCP/IP).

Los aspectos de seguridad entran en juego cuando es necesario o deseable proteger la información a transmitir de un “oponente“ que podría presentar una amenaza para la confidencialidad, autenticidad, integridad, etc.

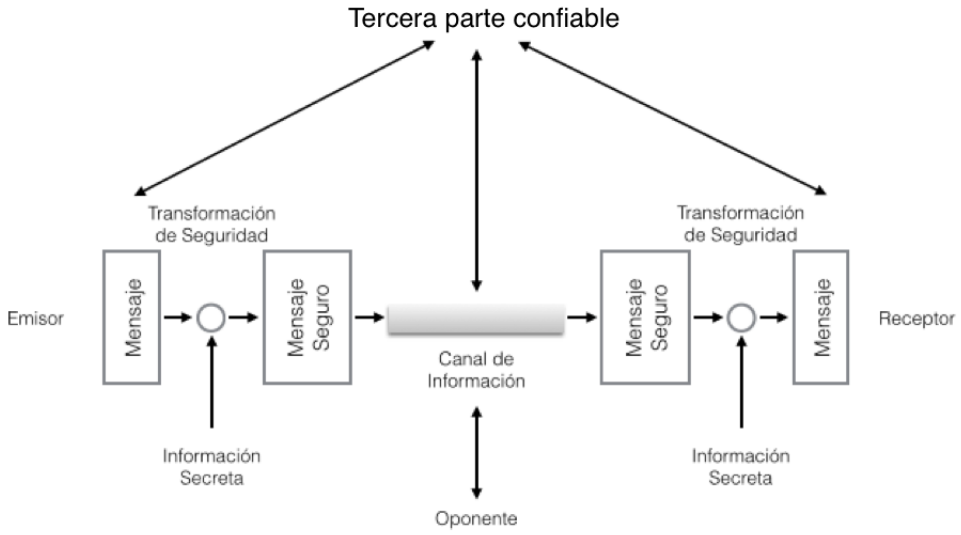

Figura 2.3: Modelo de seguridad para redes.

2.4.4. Ataques Informáticos

A continuación se detallan algunos tipos de ataques informáticos a estudiar en este trabajo.

Negación de Servicio (*Denial of Service*)

Este tipo de ataques pretenden afectar la reputación de una compañía o hacer imposible el acceso a sus servicios por parte de los clientes. También pueden ser utilizados como distracción para mantener ocupado al equipo de seguridad investigando mientras se realizan otros ataques.

Existen distintos tipos de ataques de negación de servicios [8], listados a continuación según sus nombres en inglés:

- Smurf and Fraggle
- Buffer Overflow
- Ping of death
- Teardrop
- SYN flood

Siendo este último uno de los más populares y simples de ejecutar. Su funcionamiento se basa en enviar cientos de paquetes SYN desde el nodo atacante hacia el servidor, dejando las conexiones

abiertas y consumiendo los recursos disponibles del servidor web. Con esto se logra impedir que los clientes tradicionales del servicio puedan acceder a éste como se ejemplifica en la figura 2.4.

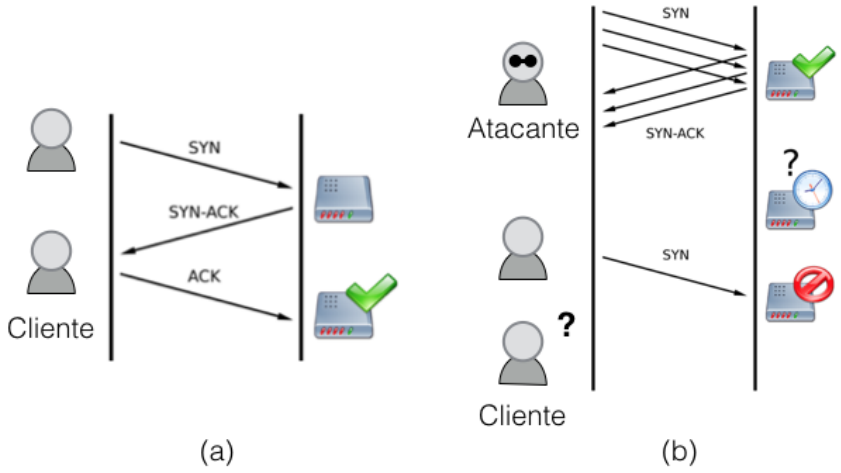

Figura 2.4: (a) *Three-way handshake* tradicional exitoso. (b) Ataque *SYN flood* impidiendo el *handshake* por parte del cliente tradicional.

SQL Injection

Los ataques de *SQL injection* funcionan de manera similar a los ataques XSS (*Cross Site Scripting*) en donde el atacante logra enviar una porción de código ejecutable a la capa lógica o de procesamiento de la aplicación web (*backend*) a través de formularios HTML desde el lado del cliente. En el caso particular de inyecciones SQL el atacante logra enviar comandos SQL a la plataforma de administración de la base de datos de la aplicación web pudiéndola mapear, alterar y controlar a discreción.

2.4.5. Contramedidas Básicas

Dada la creciente cantidad de vulnerabilidades que pueden ser explotadas hoy en día en los sistemas de comunicaciones, es importante protegerlos tomando medidas y utilizando productos adecuados según los requerimientos éstos. Dentro de las principales soluciones que usualmente se implementan para contrarrestar acciones maliciosas, están los sistemas de detección de intrusos, *firewalls* y *honeypots*, siendo los dos primeros los que se configurarán en el escenario de simulación para representar un sistema real.

Sistema de Detección de Intrusos (IDS)

La detección de intrusos es un aspecto crítico del monitoreo de redes para la seguridad, siendo esta una técnica "pasiva" ya que informa que un evento ha ocurrido pero no existe la posibilidad de prevenir el ataque o corregir el problema. Para esto existen sistemas de prevención de intrusiones (IPS), pero debido a la posibilidad de tener falsos positivos, éstos pueden ser volcados en contra de la organización si no son configurados correctamente.

El objetivo de los sistemas de detección de intrusos es analizar, evaluar y reportar actividad en la red, detectando intrusiones a partir de algoritmos que pueden ser divididos en dos categorías [13]. En primer lugar, se encuentran los basados en firmas o plantillas generadas a partir de amenazas conocidas (*misuse-based*), permitiendo comparar patrones de actividad en la red con una base de datos para encontrar similitudes y reportar la detección de ataques. Estos sistemas son eficientes a la hora de detectar ataques conocidos, sin embargo cuando aparecen nuevos métodos de intrusión son inservibles. Uno de los IDS más utilizados en la actualidad es "Snort", que utiliza la detección y reconocimiento de firmas para la detección de fallas de seguridad.

Por otro lado, se encuentran los sistemas basados en detección de anomalías (*anomaly-based*), en los cuales se caracterizan el comportamiento normal del sistema utilizando modelos de datos y técnicas de inteligencia computacional, permitiendo posteriormente al sistema detectar cuando se genera un comportamiento que escapa de la estadística normal. Uno de los problemas que tienen estos sistemas de detección es que la caracterización del estado normal puede ser muy compleja dependiendo de la red en estudio y los detectores pueden tener alta tasa de falsos positivos. Sin embargo, al ser sistemas inteligentes, tienen la capacidad de detectar ataques que utilicen nuevas metodologías de intrusión no conocidas.

Usualmente se recomienda aplicar ambos sistemas de manera complementaria para lograr mejores resultados y poder detectar fallas de seguridad de manera competente y eficaz.

Firewalls

Los sistemas de seguridad conocidos como *firewalls*, consisten en tecnologías de software o hardware que fortalecen las políticas de seguridad filtrando y fijando reglas para el tráfico que pasa a través de ellos según direcciones o puertos de origen y destino, además de los protocolos utilizados.

2.5. Inteligencia Computacional para Seguridad Informática

El objetivo principal de este trabajo es encontrar utilidad en algunas de las técnicas relacionadas con inteligencia computacional presentadas anteriormente para fortalecer servicios de seguridad informática o implementar nuevos, permitiendo predecir patrones de tráfico y detectar a tiempo posibles fallas de seguridad o ataques.

Para los sistemas tradicionales de seguridad computacional se cumple el ciclo representado en la figura 2.5, en donde, para todas las amenazas nuevas que aparecen, solo una porción es detectada y otra sección aún menor es analizada para incluir medidas de prevención en el futuro, por lo tanto, este desbalance en el ciclo permite que la sofisticación y la rapidez con que evolucionan las amenazas superen las capacidades de los sistemas de seguridad, disminuyendo finalmente su efectividad.

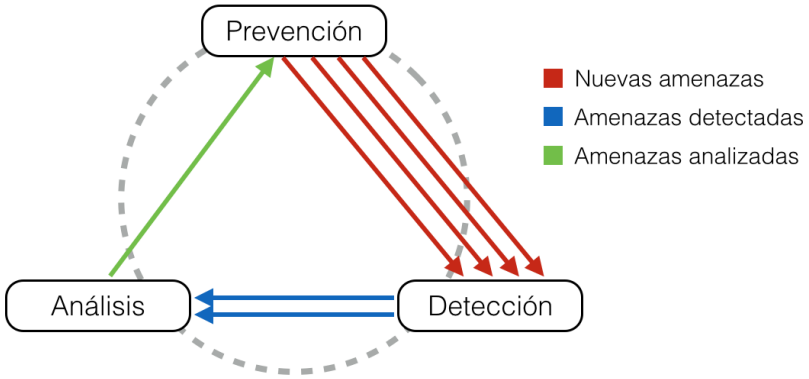

Figura 2.5: Ciclo de seguridad computacional.

Es aquí donde las técnicas de inteligencia computacional entran en juego y tienen la posibilidad de fortalecer las secciones de detección y análisis del ciclo presentado en la figura 2.5, balanceándolo, y por lo tanto haciendo más efectivos los sistemas de seguridad vigentes. Ejemplos de esto existen actualmente en la literatura, a pesar de que su desarrollo es más bien reciente y aún queda estudio por delante para establecer definitivamente el uso de esta aproximación para resolver problemas de seguridad dentro de redes de comunicaciones. Un artículo que refleja esto de excelente manera es *"Computer Security and Machine Learning: Worst Enemies or Best Friends?"*, publicado por Konrad Rieck en 2011 [14], en donde se describen las principales ventajas y desventajas de esta metodología, y que se proceden a resumir a continuación como parte de los puntos a tener en cuenta a la hora de desarrollar el presente trabajo.

Para que una solución de seguridad basada en inteligencia computacional cumpla su propósito, debe satisfacer las siguientes características:

- **Efectividad:** Cualquier método de aprendizaje aplicado a seguridad informática debe ser efectivo en la detección, análisis o prevención de amenazas. En el caso de un sistema de detección de intrusos, es obligatorio que tenga una baja tasa de falsas alarmas, pues de otro modo sería inaplicable en la provisión de un servicio.
- **Eficiencia:** Los resultados obtenidos a través del sistema basado en *machine learning* deben generarse notablemente más rápido que un sistema convencional para poder ser competitivos.
- **Transparencia:** Para un encargado de seguridad es fundamental que el sistema opere de

manera transparente, pudiendo acceder a información relevante sobre las condiciones que el sistema considera para cada una de las decisiones que toma en la detección o prevención de un ataque. De esta manera se pueden corregir posibles fallas u optimizar el funcionamiento de la solución.

- **Controlabilidad:** Esto se relaciona con el punto anterior, en donde el humano experto debería poder intervenir el sistema para corregir fallas o condiciones especiales de operación ante falsas alarmas. Por lo tanto, a pesar de que el sistema puede ser autónomo, debe permitir la supervisión activa del equipo de seguridad para mejorar los resultados.
- **Robustez:** Como cualquier extensión o bloque del sistema de seguridad, éste también se convierte en blanco de posibles ataques, por esta razón debe ser lo suficientemente robusto para evitar que atacantes afecten su proceso de aprendizaje o el proceso de detección y análisis de amenazas.

Teniendo esto en cuenta, las aplicaciones posibles para un sistema basado en inteligencia computacional son varias y con resultados favorables, permitiendo la detección proactiva de ataques [18], [15], el análisis automático de amenazas [2], [1] y la asistencia en el descubrimiento de nuevas vulnerabilidades.

Resultados similares se pretenden lograr aplicando los modelos de predicción y detección descritos en 2.1 y 2.2 para el análisis de los datos recopilados a partir de la realización controlada de los ataques descritos en 2.4.4.

Capítulo 3

Metodología

A continuación se describe la metodología y el proceso seguido para la implementación del ambiente de simulación, la realización de ataques y finalmente la recolección y el análisis de los datos.

3.1. Plan de trabajo

Antes de describir las herramientas necesarias para la realización del trabajo es necesario definir el plan de trabajo para la generación de los modelos a partir del análisis de los datos recolectados. En la figura 3.1 se muestra de manera gráfica el flujo de trabajo propuesto para obtener los resultados, en donde parte importante del desarrollo tiene que ver con la primera etapa (recolección de datos), ya que se debe implementar de manera emulada una red representativa de la arquitectura típica de un proveedor de servicios de Internet.

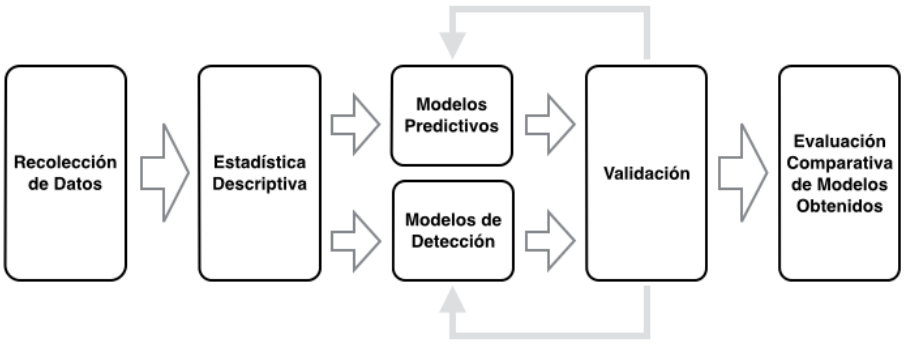

Figura 3.1: Plan de trabajo propuesto.

Una vez implementado el ambiente de simulación, y verificado su funcionamiento, se procede

a la generación de datos a partir de la recolección que hacen sondas en distintos puntos de la red para ser analizados posteriormente, se realiza la descripción correspondiente de manera estadística y se generan los modelos de predicción y detección según las técnicas descritas en el capítulo 2. Vale la pena destacar que, si bien los modelos se generan de manera independiente para la predicción y la detección, éstos pueden ser utilizados de manera complementaria para la detección efectiva de ataques. Finalmente es importante validar los modelos e iterar si es necesario para corregirlos, y comparar los resultados de detección de ataques con otras soluciones existentes, en cuanto a eficiencia y efectividad.

3.2. Herramientas

A continuación se describen las herramientas de software a utilizar para la implementación del ambiente de simulación.

Simulador de Redes GNS3

GNS3 es un simulador gráfico de redes que permite la inclusión de máquinas virtuales para la emulación de equipos reales dentro de la red. Actualmente está disponible para las tres grandes plataformas: Linux, Windows y OSX.

Virtualbox

Software de virtualización de sistemas x86 y AMD64/Intel64 disponible como software de código abierto para Windows, Linux y OSX. Permite la virtualización de equipos clientes, servidores y *(*routers) dentro del ambiente de simulación de GNS3.

Linux

Para emular los equipos clientes, *(*routers) y servidores dentro del ambiente generado en GNS3 se utilizaron distintas distribuciones de Linux según su utilidad dentro de la red y la carga de recursos que agregan al simulador completo. A continuación se listan las distribuciones y su correspondiente uso:

- **Lubuntu:** Versión liviana de Ubuntu (*Light Ubuntu*) utilizada para los equipos cliente, sonda y firewall del servidor web.
- **Debian:** Utilizada como sistema base para los equipos de ruteo. Sobre ella se instaló el software Quagga para realizar las tareas correspondientes al servicio de enrutador.
- **Kali Linux:** Utilizada en equipo atacante. Posee todas las herramientas necesarias para hacer el escaneo y explotación de vulnerabilidades dentro de la red.

- **Metasploitable:** Versión dedicada de Linux para pruebas de penetración que posee un servidor web con vulnerabilidades abiertas en sus versiones de PHP y MySQL, además de otras en servicios FTP y a nivel de sistema operativo. Permite el estudio de los ataques de manera segura dentro de una máquina virtual.

Metasploit

Metasploit es el software más usado en el mundo para las pruebas de penetración de sistemas que permite verificar vulnerabilidades y ayudar en el manejo de sistemas de seguridad. Está respaldado por una comunidad abierta y Rapid7 (www.rapid7.com), por lo que permite el descubrimiento y explotación de vulnerabilidades en un ambiente seguro dentro del simulador implementado.

Locust

Locust es una herramienta para realizar pruebas de carga de usuarios y de desempeño de servidores de fácil uso. Está diseñada especialmente para realizar pruebas de capacidad y carga en sitios web, permitiendo la programación de modelos de comportamiento de los usuarios para estudiar escenarios más realistas. Permite sólo ejecuciones con un número máximo de usuarios fijo.

FunkLoad

FunkLoad es otra herramienta disponible y programada en Python para realizar pruebas funcionales y de capacidad de carga en sitios web. A diferencia de Locust, FunkLoad permite la configuración de ciclos automáticos con cantidad máxima de usuarios simultáneos variable.

Quagga

Quagga es una *suite* de softwares de enrutamiento de código abierto que provee la implementación de los protocolos OSPF, RIP, BGP and IS-IS para plataformas basadas en Unix. Su desarrollo está basado en el proyecto GNU Zebra descontinuado en 2005. En particular para este trabajo, se instaló sobre Debian y permite la gestión de las interfaces de red de cada enrutador para efectuar el enrutamiento de manera eficiente.

IPT-Netflow

IPT-Netflow es un módulo de alto desempeño para Linux que permite exportar registros de Netflow. Tiene soporte para los protocolos v5, v9 e IPFIX de Netflow, logrando registrar tráfico

IPv4, IPv6 y eventos de traducción NAT (NEL). Para el desarrollo del trabajo, se instaló en cada uno de los (routers) para exportar los registros del flujo a una sonda común.

R

R es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se distribuye bajo la licencia GNU GPL y está disponible para los sistemas operativos Windows, OSX, Unix y GNU/Linux.

RStudio

RStudio es un ambiente de desarrollo integrado (IDE) que funciona como una poderosa interfaz de usuario para R. Es la principal herramienta a utilizar para el análisis de los datos recolectados y la generación de los modelos. Está disponible para Windows, Linux y OSX.

3.3. Implementación del Ambiente de Simulación

Con todas las herramientas descritas anteriormente, se procedió a construir la red emulada que representa la arquitectura básica típica de un de un proveedor de servicios de Internet (ISP). En la figura 3.2 se muestra la distribución de los distintos equipos, como máquinas virtuales, dentro del ambiente de simulación en GNS3, en donde los puntos etiquetados como e0, e1, e2 y e3 corresponden a puertos *Ethernet* de cada una.

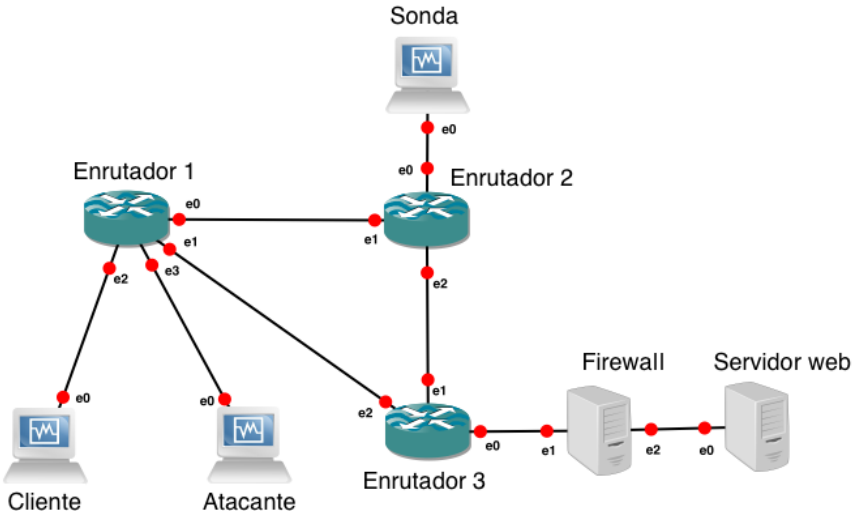

Figura 3.2: Figura extraída desde GNS3 con la arquitectura simulada.

La idea detrás de este diseño fue la de incluir todos los componentes más importantes de la arquitectura para lograr simulaciones lo más cercanas posible al comportamiento de un sistema real, tomando en cuenta las limitaciones de hardware y software con las que se contó para realizar el trabajo.

Modelo	**Macbook Pro mid 2012**
Procesador	Intel Core i5 2.5 GHz
Memoria RAM	16 GB 1600 MHz DDR3
Disco Duro	500 GB SATA
Sistema Operativo	OSX 10.11.1

Tabla 3.1: Características del computador utilizado para desarrollar la totalidad del trabajo.

3.3.1. Asignación de Software

A continuación se muestra la lista de software específico asignado para cada máquina virtual del ambiente de simulación implementado. Cada componente fue seleccionado teniendo en cuenta su funcionalidad, facilidad de uso y recursos disponibles en la máquina anfitriona.

Máquina	**Sistema Operativo**	**Aplicaciones adicionales**
Cliente tradicional	Lubuntu 14.10	Generación de tráfico estático: · Locust Generación de tráfico dinámico: · FunkLoad
Cliente malicioso	Kali Linux 1.1.0	—-
Routers	Debian 7.8	Aplicación de enrutamiento: · Quagga 0.99.24.1 Aplicación para Netflow: · ipt-Netflow 2.1
Firewall	Lubuntu 14.10	Firewall: · iptables
Servidor Web	Metasploitable Linux 2.0	—-
Sonda	Lubuntu 14.10	Captura Netflow: · nfcapd · nfdump Análisis de datos: · RStudio

Tabla 3.2: Lista de asignación de software del sistema diseñado.

3.3.2. Asignación de Direcciones IP

Para la asignación de direcciones IP, se evitó montar un servidor DHCP optando finalmente por utilizar el método manual para configurar cada uno de los puertos *ethernet* del sistema. Si bien esto requiere más tiempo de configuración, permite tener una comprensión más acabada de la topología a nivel de red del sistema. A continuación se lista el detalle de las direcciones asignadas a cada puerto, donde se sigue la siguiente lógica de asignación:

- Red local de clientes: 10.0.10.x
- Red local de sonda: 10.0.20.x
- Red local de servidor: 10.0.30.x
- Red privada de servidor: 192.168.0.x
- Red troncal 1: 10.0.1.x
- Red troncal 2: 10.0.2.x
- Red troncal 3: 10.0.3.x

Máquina	**Puerto**	**Dirección IP**
Cliente	eth0	10.0.10.10
Atacante	eth0	10.0.10.20
(Router) 1	eth0	10.0.1.1
	eth1	10.0.2.1
	eth2	10.0.10.1
	eth3	10.0.10.2
(Router) 2	eth0	10.0.20.1
	eth1	10.0.1.1
	eth2	10.0.3.1
(Router) 3	eth0	10.0.30.1
	eth1	10.0.3.2
	eth2	10.0.2.2
Sonda	eth0	10.0.20.10
Firewall	eth0	10.0.30.10
	eth1	192.168.0.10
Servidor web	eth0	192.168.0.20

Tabla 3.3: Asignación de direcciones IP para cada máquina virtual.

3.4. Generación de Tráfico Web

La primera parte importante del trabajo realizado tiene relación con la generación de tráfico web que simule el comportamiento de un número determinado de clientes accediendo a un sitio,

permitiendo posteriormente extraer información al respecto desde los *(*routers) y poder caracterizar los estados de operación normal y bajo ataque. Para esto fue necesario desarrollar un modelo del comportamiento de un usuario al navegar a través de una página web, que permitiera ser replicado con la ayuda de los *frameworks* descritos en la sección 3.2 y así lograr el tráfico requerido para el estudio.

3.4.1. Modelo de Comportamiento de un Cliente Web

El modelo diseñado para simular el comportamiento de un usuario consideró la posibilidad de recorrer la totalidad del árbol que conforma la página web alojada en el servidor y además incluir la variabilidad del tiempo de permanencia en cada vista que podría tener cada usuario de manera aleatoria. En la figura 3.3 se muestra el esquema básico del modelo implementado y que es equivalente para ambos *frameworks* utilizados. El código correspondiente a cada implementación se encuentra anexo en el apéndice B.

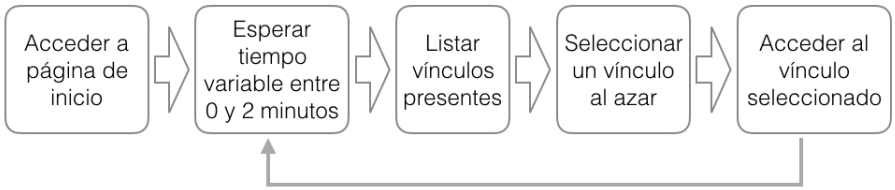

Figura 3.3: Modelo simplificado del comportamiento de un usuario al navegar a través de una página web.

3.4.2. Tráfico con Cantidad Estática de Usuarios Utilizando Locust

Para generar la carga de tráfico base es necesario ejecutar Locust con el *script* previamente programado que implementa el modelo de usuario propuesto. Esto se logra siguiendo los siguientes pasos:

1. Iniciar sesión en la máquina asociada al cliente tradicional.
2. Abrir una ventana de terminal, ir a la carpeta en donde se encuentra almacenado el *script* de comportamiento de usuario para Locust y ejecutar el siguiente comando:

   ```
   $ locust -f test\_Base.py
   ```

   donde “test_Base.py” es el nombre asignado al *script* desarrollado e incluido como anexo a este documento.
3. Luego, se debe acceder a través del navegador web a la dirección **http://127.0.0.1:8089/** para acceder a la interfaz gráfica de Locust.

4. Seleccionar la cantidad máxima de usuarios a simular y la tasa de inicio de las instancias asociadas a éstos (*hatch rate*, medido en cantidad de usuarios iniciados por segundo), como se muestra en la figura 3.4.

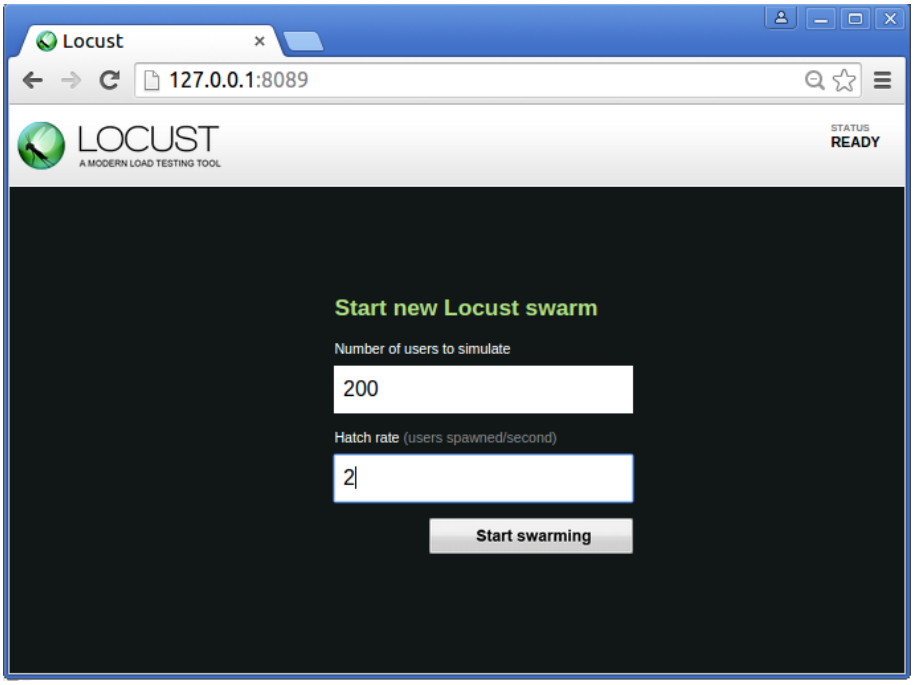

Figura 3.4: Interfaz gráfica de Locust para configuración y manejo del generador de tráfico basal.

5. Presionar “Start Swarming”.
6. Una vez terminada la simulación se puede presionar el boton “Stop” de la interfaz de Locust para terminar las instancias iniciadas.

3.4.3. Tráfico con Cantidad Dinámica de Usuarios Utilizando FunkLoad

Para generar la carga de tráfico dinámica es necesario ejecutar Funkload con el *script* previamente programado que implementa el modelo de usuario propuesto. Esto se logra siguiendo los siguientes pasos:

1. Iniciar sesión en la máquina asociada al cliente tradicional.
2. Abrir una ventana de terminal, ir a la carpeta en donde se encuentra almacenado el *script* de comportamiento de usuario para Funkload y ejecutar el siguiente comando:

```
$ fl-run-bench test_Dinamico.py Dinamico.test_dinamico
```

donde “test_Dinamico.py” es el nombre asignado al *script* desarrollado e incluído como anexo a este documento.

3.5. Ejecución de Ataques

Otra parte importante del estudio realizado tiene que ver con los ataques informáticos ejecutados durante las simulaciones. Durante el desarrollo inicial del trabajo se probaron distintos tipos de ataques decidiendo finalmente por utilizar como ataque principal el de denegación de servicio conocido como "SYN-flood" (descrito en la sección 2.4.4) por su amplia documentación, popularidad y facilidad de ejecución utilizando las herramientas descritas en la sección 3.2.

Para cada ejecución de un ataque "SYN-flood" fue necesario seguir la secuencia de pasos descrita a continuación utilizando "Metasploit" como herramienta principal.

1. Iniciar sesión en la máquina atacante (Kali Linux).
2. Abrir "Metasploit" dentro de una ventana de terminal.

```
$ msfconsole
```

3. Iniciar programa auxiliar para ataque SYN-flood.

```
msf > use auxiliary/dos/tcp/synflood
```

4. Configurar ataque, asignando dirección IP y puerto de destino. En particular para este trabajo, la dirección del servidor web es 10.0.30.10:80.

```
msf > set rhost 10.0.30.10
```

5. Ejecutar ataque.

```
msf > exploit
```

6. Finalmente, luego de esperar el tiempo necesario para la recolección de datos, se debe terminar el programa para volver al estado normal de operación.

```
msf > exit
```

3.6. Recolección de Datos

La recolección de los datos se realizó en base a registros de los flujos de datos a nivel de la capa IP, incluyendo información de tiempos de inicio y término de los flujos, dirección IP y puertos de origen y destino, protocolo, cantidad de paquetes, cantidad de bytes, etc.

Este proceso consta de dos partes, la recolección local en cada *router* utilizando ipt-Netflow y el reenvío de esta información a la máquina dedicada como sonda. Por lo tanto es necesario iniciar programas dedicados en ambos extremos para una recolección de datos exitosa. A continuación se describe el proceso completo para iniciar la recolección de datos en cada simulación.

3.6.1. Recolección vía Ipt-Netflow

Antes de iniciar cada una de las simulaciones de tráfico y ataques DoS se ejecutó en todos los *(*routers) la secuencia de inicio del monitor de flujo "ipt-Netflow", permitiendo el almacenamiento local de los registros y su reenvío a la sonda. Para esto se debe iniciar sesión en cada uno de los sistemas asociados a *(*routers) y ejecutar el siguiente comando:

```
$ modprobe ipt_NETFLOW destination=10.0.20.10:XXXX
  sampler=deterministic:1 maxflows=0
```

Donde **XXXX** corresponde al número de puerto en el cual la sonda recibirá la información recolectada (puertos 2055, 2056, 2057, para los *(*routers) 1, 2 y 3 respectivamente).

3.6.2. Captura de Registros en Sonda vía Nfcap

Para capturar en la sonda los registros enviados por cada uno de los *(*routers), es necesario iniciar tres instancias de recolección utilizando "nfcap". Para esto se deben seguir los siguientes pasos:

1. Iniciar sesión en la máquina asociada a la sonda.
2. Abrir tres ventanas de terminal y en cada una de ellas ejecutar el siguiente comando:

   ```
   $ nfcapd -w -T all -p XXXX -I 'routerN'
     -l /home/host1/Analisis/Netflow/routerN -s 2
   ```

   en donde XXXX corresponde a los puertos 2055, 2056 y 2057 para recibir la información de los *(*routers) 1, 2 y 3 respectivamente, y donde 'routerN' corresponde al nombre seleccionado para identificarlos.
3. Luego, una vez terminada la simulación, se debe terminar el proceso de recolección en las tres instancias iniciadas, presionando *Ctrl+C* o simplemente cerrando las ventanas asociadas. Los registros quedarán almacenados en bruto en la dirección especificada en el comando anterior.

3.6.3. Almacenamiento Definitivo en Formato CSV

Finalmente para el proceso de almacenamiento de los registros, se utilizaron archivos de valores separados por coma (.csv) extraídos desde los archivos en bruto generados por el capturador de flujos IP. La ventaja de utilizar archivos CSV es que son fácilmente manejados e importados a RStudio. Para esto se utilizó la herramienta "nfdump" disponible a través del terminal de la máquina virtual asociada a la sonda, siguiendo los siguientes pasos:

1. Iniciar sesión en la máquina asociada a la sonda.

2. Abrir una ventana de terminal y ejecutar la siguiente secuencia para compilar la información correspondiente de cada enrutador .csv independientes:

```
$ nfdump -R /home/host1/Netflow/routerN -o csv >>
  /media/host1/Dumps/routerN.csv
```

donde "routerN" puede ser nuevamente un identificador seleccionado para cada enrutador.

3.7. Procedimiento para Ejecución de Simulación

Para la ejecución de cada una de las simulaciones se sigue una secuencia de tareas similar para ambos estados de operación estudiados. A continuación se listan de manera resumida los pasos a seguir para lograr una simulación exitosa.

3.7.1. Simulación para Tráfico en Estado de Operación Normal y Bajo Ataque de Denegación de Servicio

Antes de comenzar una simulación es necesario verificar que todos los sistemas están correctamente configurados y que la red emulada funcionará sin problemas. Luego de la verificación inicial se procede a ejecutar la simulación según la siguiente lista de pasos:

1. Abrir GNS3 y seleccionar el proyecto asociado al ambiente de simulación implementado.
2. Presionar botón "Play" ubicado en la barra de herramientas superior. Luego de esto se debe esperar a que todos los sistemas inicien sesión e indiquen su conexión exitosa dentro de GNS3.
3. Iniciar sistema de recolección y reenvío de registros Netflow en cada uno de los *(*routers) como se indica en la sección 3.6.1.
   Nota: Por comodidad, fue incluido en los (*routers) un script que inicia la recolección automáticamente al encenderlos, por lo tanto este paso puede ser obviado a menos que existan problemas de recolección en el sistema.*
4. Iniciar la captura de registros de Netflow en la máquina dedicada como sonda según lo descrito en la sección 3.6.2.
5. Iniciar generador de tráfico con cantidad de usuarios estática (carga base) según lo descrito en la sección 3.4.2.
6. Iniciar generador de tráfico con cantidad de usuarios variable (carga dinámica) según lo descrito en la sección 3.4.3.
7. Ejecutar ataques según lo descrito en la sección 3.5.
8. Terminar generadores de tráfico y recolectores de información.

9. Terminar la sesión en GNS3 presionando el botón “Stop” ubicado en la barra de herramientas superior.

3.7.2. Simulaciones Realizadas

Como parte del estudio, y dependiendo del objetivo buscado, se realizaron múltiples simulaciones con las siguientes características:

- Simulación con cantidad fija de 500 usuarios para probar funcionamiento del *framework* Locust.
- Simulación con ciclos de cantidad variable entre 200 y 500 usuarios para probar funcionamiento de *framework* FunkLoad.
- Simulación de prueba con ciclos de 0 a 1500 usuarios, por escalones de 100 clientes para encontrar experimentalmente el límite de operación de la máquina virtual asociada al servidor y verificar la recolección de datos.
- Simulación sin tráfico web generado y con ataques de denegación de servicio para probar recopilación de datos durante los ataques.
- Simulación definitiva para caracterización de estado de operación normal, generando tráfico basal constante equivalente a 200 usuarios utilizando Locust y ciclos de 400 y 800 usuarios utilizando FunkLoad.
- Simulación definitiva para generar serie de prueba con parámetros de generación de tráfico idénticos a la anterior, pero esta vez ejecutando ataques de manera aleatoria para probar modelos de detección.

3.8. Procesamiento de Datos Recolectados

3.8.1. Metodología de Análisis de Datos

La metodología utilizada para la obtención de resultados se basa principalmente en el uso de herramientas de estadística para la caracterización del comportamiento del flujo de datos a través de los *(*routers) y así posteriormente obtener modelos de predicción de tráfico y detección de ataques de denegación de servicio.

De esta manera, en primera instancia se generan gráficos de distintos tipos para tener una visión general de lo que ocurre con el sistema en operación normal y bajo ataque. Luego, trabajando con las series de tiempo para el tráfico agregado por minuto en la subida y la bajada del servidor, se analizan los residuos del ajuste lineal calculado para estudiar las funciones de autocorrelación y autocorrelación parcial (ACF y PACF por sus siglas en inglés). Con esto se busca tener una primera

aproximación de las características que debería cumplir el modelo ARIMA o SARIMA generado posteriormente, en cuanto a los valores máximos a considerar para sus parámetros p, q y d.

Con los resultados de este análisis se procede a generar modelos con todas las combinaciones posibles de parámetros (p,d,q) para encontrar el que mejor se ajuste a los datos, utilizando como criterio principal el valor de AIC asociado a cada uno. Finalmente con el modelo seleccionado se genera una predicción y se evalúa en cuanto a su precisión y utilidad para ser usado como mecanismo complementario de detección de ataques de denegación de servicio.

Por otro lado, para la detección de ataques utilizando la divergencia de Kullback-Leibler, se caracteriza el estado de operación normal mediante una distribución de probabilidad empírica para los niveles de tráfico, que luego sirve como base de comparación para el cálculo de la divergencia de Kullback-Leibler asociada a una ventana móvil que recorre la serie de tiempo de prueba. Luego, con el valor de la divergencia en cada instante, se evalúa establecer un umbral que permita decidir si la ventana en estudio corresponde a un caso de estado de operación normal o un estado bajo ataque de denegación de servicio.

El procesamiento y análisis de los datos recolectados a partir de las simulaciones fue realizado utilizando *scripts* en R exclusivamente desarrollados para cada uno de los objetivos del trabajo. De esta manera se tienen tres archivos .R que permiten generar los gráficos y cálculos presentados en el capítulo 4 de resultados.
Para el detalle de la implementación se puede revisar el código correspondiente a cada uno de los archivos descritos en el apéndice A.

3.8.2. Lista de Librerías Adicionales para R Instaladas

- **xts** [16]: Paquete para el manejo simple de las distintas clases de datos basados en tiempo dentro de R.
- **forecast** [9]: Métodos y herramientas para el análisis de series de tiempo univariadas.
- **entropy** [10]: Implementa múltiples estimadores para la entropía. Además provee una función para el cálculo de la divergencia de Kullback-Leibler a partir de datos experimentales.
- **lmtest** [23]: Colección de pruebas para modelos de regresión lineal.
- **qmao** [17]: Incluye funciones para manejo de series de tiempo, en particular, una para convertirlas en estrictamente regulares llamada "*MakeStrictlyRegular*".
- **astsa** [20]: Paquete que incluye *scripts* para el análisis de series de tiempo y generación de modelos ARIMA.
- **HistogramTools** [21]: Funciones utilitarias para el manejo de histogramas.

3.8.3. Estadística Descriptiva y Modelo de Predicción de Tráfico

En primer lugar se tiene el archivo "Modelo_Prediccion.r" en donde se generan cálculos y gráficos de estadística descriptiva, además de los cálculos asociados a la obtención del modelo de predicción de tráfico. A continuación se detallan todos los pasos que sigue el *script* para lograr el objetivo:

1. Importar archivos .csv a estudiar. Estos archivos corresponden a los de simulación de tráfico en estado de operación normal y de simulación de prueba para los modelos de predicción de tráfico y detección de ataques.
2. Preprocesamiento de datos. Extracción de columnas útiles y creación de *data frames* a utilizar.
3. Separación de flujos de entrada y salida del servidor y formación de series de tiempo de tráfico agregado por minuto.
4. Generación de gráficos de estadística descriptiva. Serie de tiempo, histogramas y *boxplots*.
5. Cálculo de modelo de predicción de tráfico.
   a) Dividir serie de tiempo en porciones de entrenamiento y validación.
   b) Revisión de ajuste lineal y residuos.
   c) Cálculo de la función de autocorrelación (ACF) y función de autocorrelación parcial (PACF) para serie de tiempo original.
   d) Análisis de ACF y PACF para serie de tiempo diferenciada.
   e) Iteración para encontrar el modelo con menor valor para el criterio de información de Akaike (AIC).
6. Prueba de predicción utilizando modelo obtenido.

3.8.4. Detección Utilizando Divergencia de Kullback-Leibler

A continuación se tiene el archivo "Detección_KL.r" que basándose en los archivos importados y procesados por el *script* anterior caracteriza el estado de operación de la red en estado normal de operación y calcula la divergencia de Kullback-Leibler para efectuar la detección de un ataque de denegación de servicio. Los pasos que implemente este *script* son:

1. Calcular histogramas personalizados como caracterización de distribución de probabilidad empírica para los niveles de tráfico de subida y bajada en estado de operación normal.
2. Iteración de cálculo de divergencia de Kullback-Leibler entre ventana móvil que recorre la serie de prueba y la caracterización lograda en el punto anterior.
3. Evaluar umbral de detección.

Capítulo 4

Resultados y Discusión

A continuación se presentan los resultados más importantes obtenidos durante el desarrollo del trabajo asociado a las simulaciones y análisis de datos generados. Todos los resultados presentados a continuación corresponden a los extraídos desde el *router* número tres, ya que manteniendo la premisa de realizar la extracción de datos y el análisis exclusivamente desde la arquitectura del ISP, éste presenta los mejores resultados por su cercanía al servidor web.

Los parámetros de la simulación utilizada para obtener lo presentado a continuación corresponde a una con los siguientes parámetros:

- 200 usuarios de carga base constante durante el total de la simulación.
- Períodos alternados de aproximadamente 1,5 horas con 0, 400 y 800 usuarios adicionales a los 200 usuarios de carga constante, correspondientes a la carga dinámica.
- Duración total de la simulación: 14 horas.

4.1. Estadística Descriptiva

4.1.1. Series de Tiempo

Para comenzar se presentan en las figuras 4.1 y 4.2 las series de tiempo utilizadas para el análisis, correspondientes a la simulación específicada anteriormente en el caso del tráfico de entrada y salida del servidor web agregado por minuto.

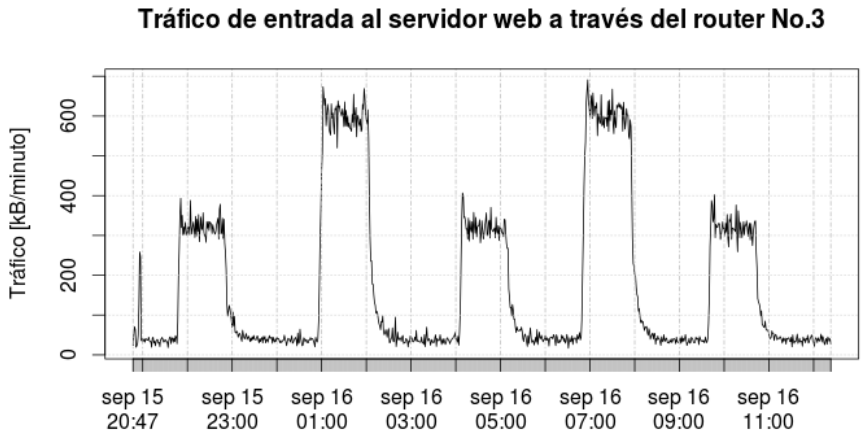

Figura 4.1: Serie de tiempo original para el tráfico de entrada al servidor web a través del *router* número 3.

Para el caso de la subida (figura 4.1) se pueden ver claramente los tres niveles de tráfico generados (bajo, medio y alto), equivalentes aproximadamente a flujos de 50 kB/minuto, 300 kB/minuto y 600 kB/minuto respectivamente. Lo bajo de estos niveles se debe a que este tráfico corresponde principalmente a solicitudes al servidor web, por lo tanto son flujos de poca cantidad de paquetes en dirección de subida.

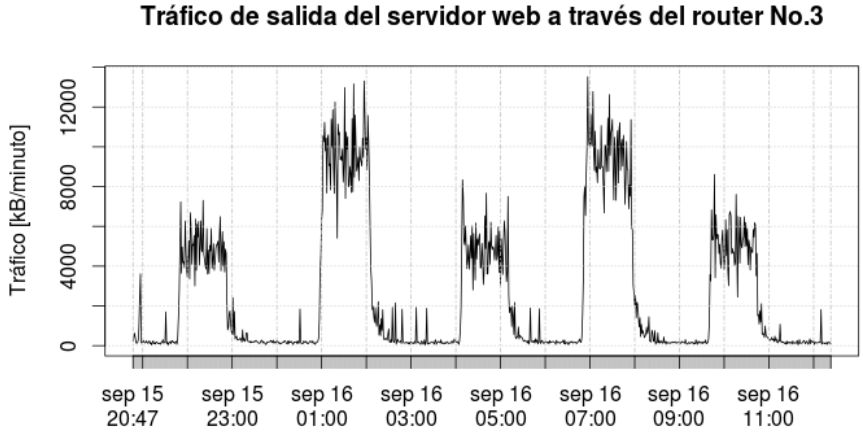

Figura 4.2: Serie de tiempo original para el tráfico de salida del servidor web a través del *router* número 3.

Un patrón similar se puede ver en la 4.2 para el caso de la bajada, pero esta vez con niveles

de tráfico hasta 16 veces más grande. Este comportamiento coincide con lo esperado para el tráfico web tradicional, que es asimétrico por la naturaleza de consumo de contenido de los usuarios.

4.1.2. Resumen Estadístico

A continuación se presentan las tablas que resumen la estadística básica de las series utilizadas en el análisis posterior. Se incluyen valores para la media, desviación estándar, coeficiente de variación y cantidad de datos, además del detalle de los cuartiles asociados a cada caso.

Serie	**Media [kB/minuto]**	**Desv. Estándar**	**Coef. Var.** $(\frac{\sigma}{\bar{X}})$	**Nº de datos**
Subida	186,4	201,7	1,1	911
Bajada	2.681,6	3.465,3	1,2	911

Tabla 4.1: Tabla resumen de estadísticas para las series en estado de operación normal.

Serie	**Mínimo**	**25 %**	**50 %**	**75 %**	**Máximo**
Subida [kB/minuto]	16,8	38,1	53,1	317,3	691,0
Bajada [kB/minuto]	26,9	171,3	339,8	4.909,3	13.517,7

Tabla 4.2: Tabla de cuartiles para la tasa de transferencia en kilobytes por minuto para las series en estado de operación normal.

De este primer resumen se rescata la asimetría de transferencia de datos para el tráfico de subida en comparación con el de bajada, llegando a valores máximos un orden de magnitud más grandes para el caso de este último.

Serie	**Media [kB/minuto]**	**Desv. Estándar**	**Coef. Var.**	**Kurtosis**	**Nº de datos**
Subida	329,8	396,8	1,2	3,9	366
Bajada	3.126,5	3.749,0	1,2	0,2	366

Tabla 4.3: Tabla resumen de estadísticas para las series de prueba que incluyen ataques de denegación de servicio.

Serie	**0 %**	**25 %**	**50 %**	**75 %**	**100 %**
Subida	12,7	40,7	257,4	545,2	1.956,2
Bajada	26,9	163,2	508,4	5.762,3	13.679,8

Tabla 4.4: Tabla de cuartiles para la tasa de transferencia en kilobytes por minuto para las series de prueba que incluyen ataques de denegación de servicio.

Algo similar se puede notar para las series de prueba que incluyen ataques de denegación de servicio (ver tablas 4.3 y 4.4), pero la anomalía generada por los ataques genera un alza en la media para el tráfico de subida y los valores máximos asociados a éste. En particular se puede ver que llega a ser hasta tres veces más alto que en estado de operación normal.

4.1.3. Histogramas

Para entender mejor las características del tráfico en ambos sentidos se generaron los histogramas presentados en las figuras 4.3 y 4.4.

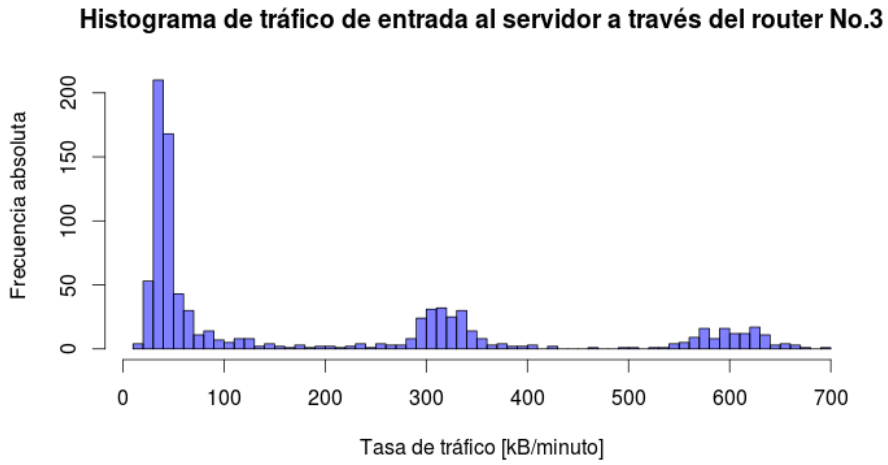

Figura 4.3: Histograma para el tráfico de entrada al servidor web a través del *router* número 3.

En la figura 4.3, para el caso de la subida, se puede ver mejor la distribución de los niveles de tráfico con los tres niveles que cualitativamente presentan acumulaciones centradas donde corresponde.

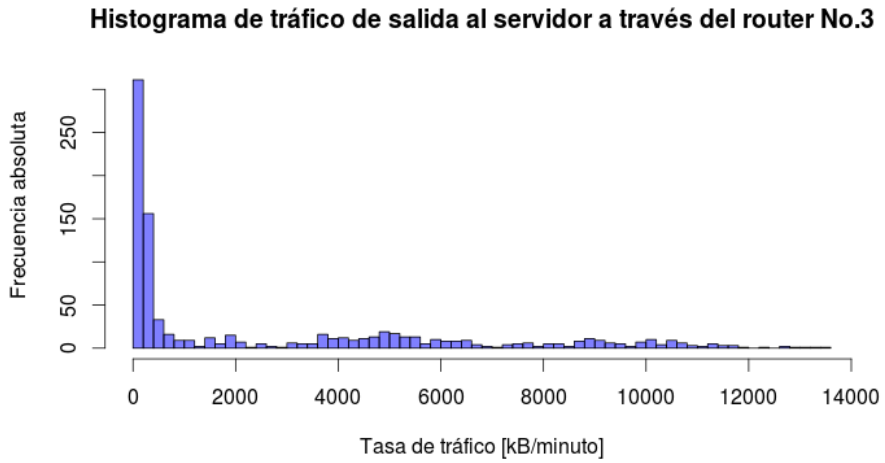

Figura 4.4: Histograma para el tráfico de salida del servidor web a través del *router* número 3.

Para el caso de la bajada (figura 4.4), al ser valores de flujo menos estables (mayor varianza) los tres niveles de tráfico no son tan notorios como en el caso anterior. Además se puede ver claramente el efecto de la asimetría de tráfico al abarcar un rango mucho más amplio.

4.1.4. Boxplots

Otra forma de entender la asimetría entre el tráfico de entrada y el de salida del servidor web es utilizando un *boxplot*.

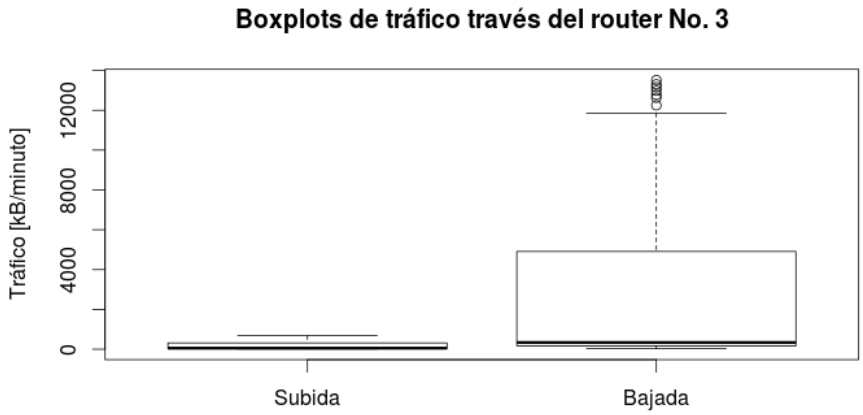

Figura 4.5: Boxplot comparativo para el tráfico a través del *router* número 3.

En la figura 4.5 se aprecia cómo el tráfico de bajada cubre mayor rango, presenta en general mayor varianza y está menos localizado.

4.2. Modelo de Predicción de Tráfico

Desde aquí en adelante, todos los resultados presentados para el modelo de predicción corresponden al trabajo realizado con la serie de tiempo para el tráfico de entrada al servidor web. Entonces, como primer paso y siguiendo la metodología descrita en la sección anterior, es necesario analizar la función de autocorrelación y autocorrelación parcial [6] para dicha serie.

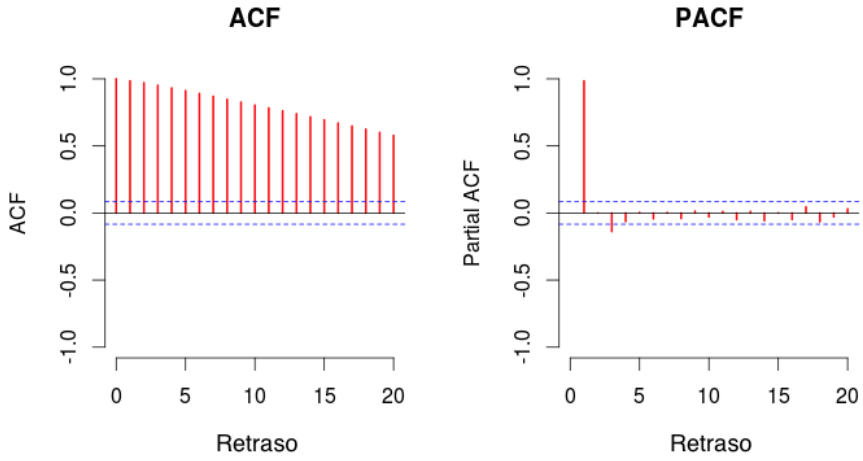

Figura 4.6: Función de autocorrelación y autocorrelación parcial para la serie de tiempo de tráfico de entrada al servidor web.

De esta manera, en la figura 4.6 se pueden ver dos efectos clave para el análisis que viene. En el caso de la función de autocorrelación, se ve un decaimiento suave en su valor, mientras que en la función de autocorrelación parcial se ve una fuerte caída a partir del segundo retraso. Según lo investigado [4], esto indicaría que los valores altos de la función de autocorrelación para retrasos mayores o iguales que 2, se deberían netamente a la ”propagación”de la autocorrelación para el primer retraso. Esto quiere decir que, por las características de la serie en estudio, los retrasos mayores a 2 tienen alta correlación con el retraso de un paso, por lo tanto es ésta autocorrelación la que domina el comportamiento de las demás. Además esto mostraría que el modelo a desarrollar debería tener una fuerte componente autoregresiva.

Luego de esto es necesario realizar el análisis nuevamente con la serie diferenciada una vez con respecto al tiempo, para hacerla estacionaria y encontrar mayor información con respecto al posible modelo a utilizar.

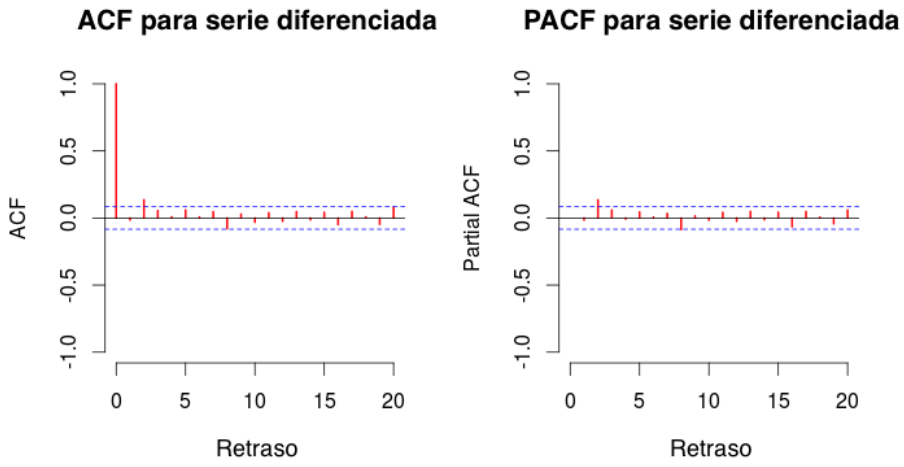

Figura 4.7: Función de autocorrelación y autocorrelación parcial para la serie de tiempo de tráfico de entrada al servidor web diferenciada una vez.

En la figura 4.7 se puede ver el resultado de este análisis y se confirma que el modelo debería tener un valor de $d = 1$ y un componente AR no mayor a 2.

4.2.1. Modelo de Predicción

Al realizar las iteraciones para encontrar el mejor modelo, se verificó que en efecto el que mejor se ajusta es uno ARIMA(2,1,0) con las siguientes características:

	Coeficiente	
	AR1	**AR2**
Valor calculado	0.0053	0.1284
Desv. estándar	0.0521	0.0520

AIC	3604.79

Tabla 4.5: Características de modelo de predicción seleccionado.

donde $AR1$ corresponde al factor ϕ_1 y $AR2$ al factor ϕ_2 en la expresión para un modelo ARIMA presentada en la ecuación 2.5 de la sección 2.1.2. Por lo tanto la expresión analítica para el modelo esta dada por:

$$(1 - \sum_{k=1}^{2} \phi_k B^k)(1 - B)^1 Z_t = e_t \tag{4.1}$$

que es equivalente a escribir el valor para la serie Zt de la siguiente manera:

$$Z_t = (1-\phi_1)Z_t - (\phi_1-\phi_2)Z_{t-1} - \phi_2 Z_{t-3} + e_t$$
$$Z_t = 0,9947 \cdot Z_{t-1} + 0,1231 \cdot Z_{t-2} - 0,1284 \cdot Z_{t-3} + e_t$$

Luego, como ejemplo de predicción se realizaron pruebas de pronóstico de tráfico, obteniéndose resultados similares a lo que se muestra a continuación.

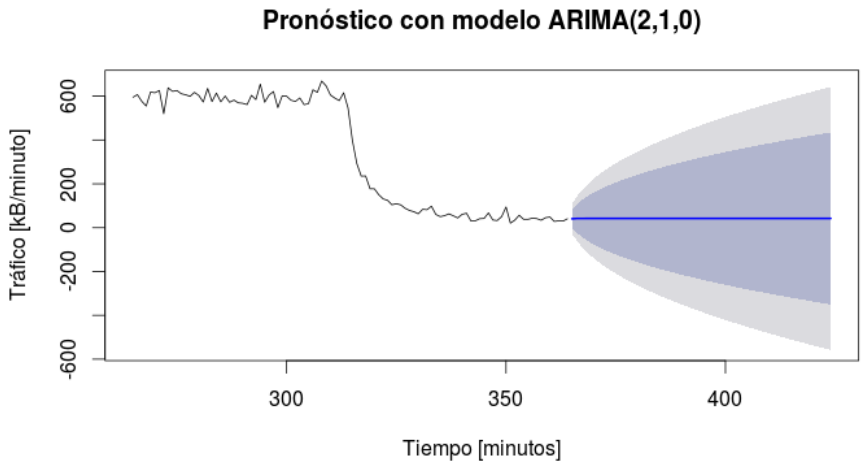

Figura 4.8: Pronóstico utilizando el modelo ARIMA(2,1,0) obtenido.

En la figura 4.8 se puede ver el resultado de predicción con su respectivo intervalo de confianza representado en azul para el 80 % y gris para el 95 %.

A partir de esto se estima que el modelo tiene buena precisión de predicción para pocos pasos hacia adelante, sin embargo para su utilización como herramienta complementaria de predicción de ataques de denegación de servicio, esto no debería proponer mayores problemas. Ésto debido a que en general el nivel de tráfico de subida en estado de ataque corresponde a varias veces el nivel máximo medido en estado de operación normal, por lo tanto se podría asegurar que su valor estará muy por fuera del intervalo de confianza de la predicción y por lo tanto podría ser detectado sin problemas.

4.3. Modelo de Detección de Ataques de Denegación de Servicio

A continuación se presentan los resultados obtenidos para el desarrollo de la detección de ataques de denegación de servicio en base al cálculo de la divergencia de Kullback-Leibler,

utilizando distribuciones empíricas de probabilidad (histogramas).

4.3.1. Serie de Prueba

La serie de prueba a utilizar corresponde a la mostrada en las figuras 4.9 y 4.10, en donde se representa con fondo rojo los momentos en que se efectuó un ataque de denegación de servicio ”SYN flood”.

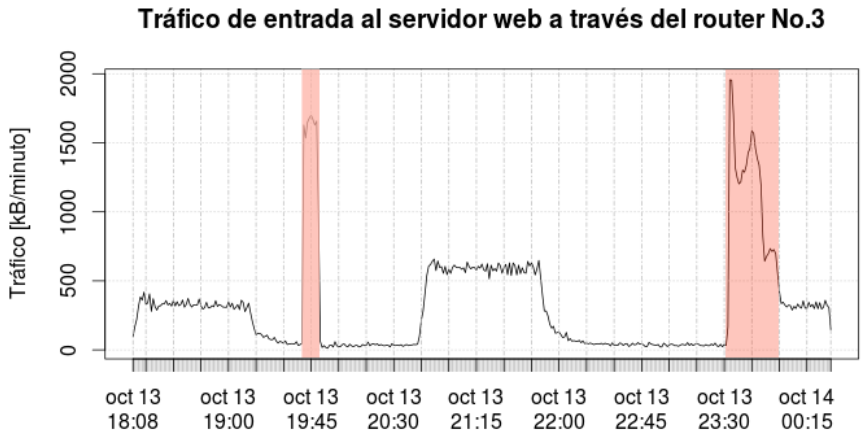

Figura 4.9: Serie de tiempo de prueba para tráfico de entrada al servidor web.

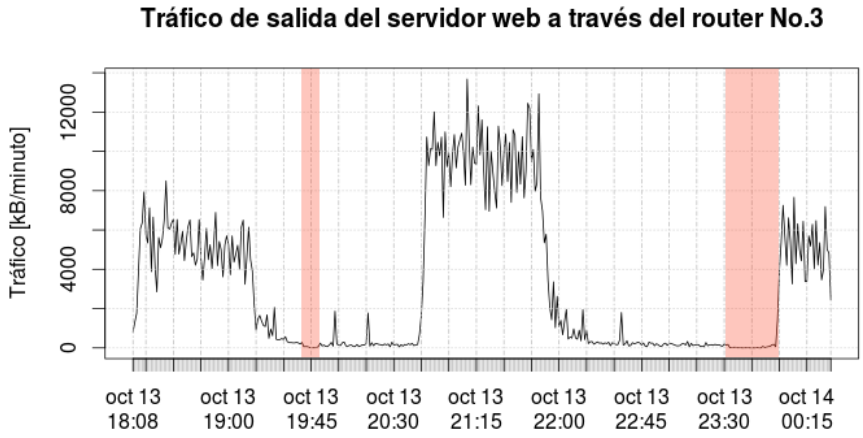

Figura 4.10: Serie de tiempo de prueba para tráfico de salida del servidor web.

A simple vista se aprecian las alzas de tráfico de subida esperadas por la avalancha de paquetes ”SYN”generados por el ataque (figura 4.9) y las caídas a cero en el tráfico de bajada (figura 4.10).

A partir de esta primera comparación se decidió seguir el análisis utilizando solo la serie de tráfico de subida pues presentó cualitativamente mejores posibilidades de éxito en la detección por la gran diferencia observada en el tráfico al comparar los instantes de estado de operación normal y bajo ataque.

4.3.2. Caracterización de Estado de Operación Normal

Para comenzar se caracterizó el estado de operación normal, en cuanto a tráfico por minuto, con un histograma con cajones de 100 kB/minuto de ancho.

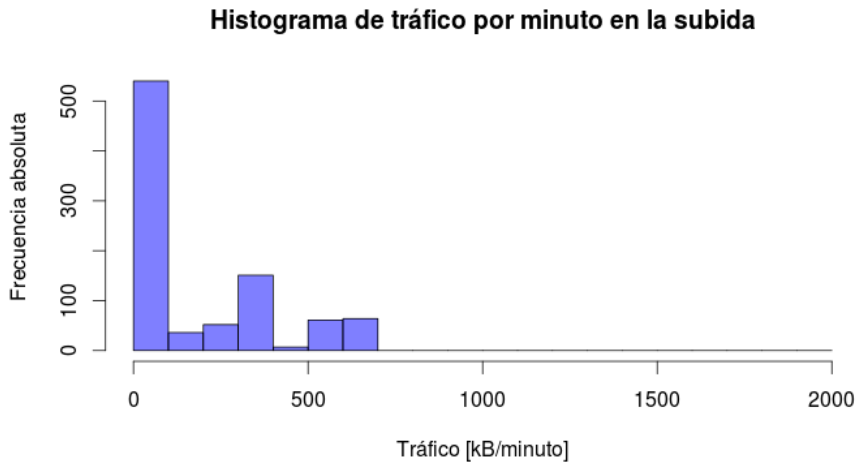

Figura 4.11: Histograma de caracterización de tráfico de entrada al servidor web.

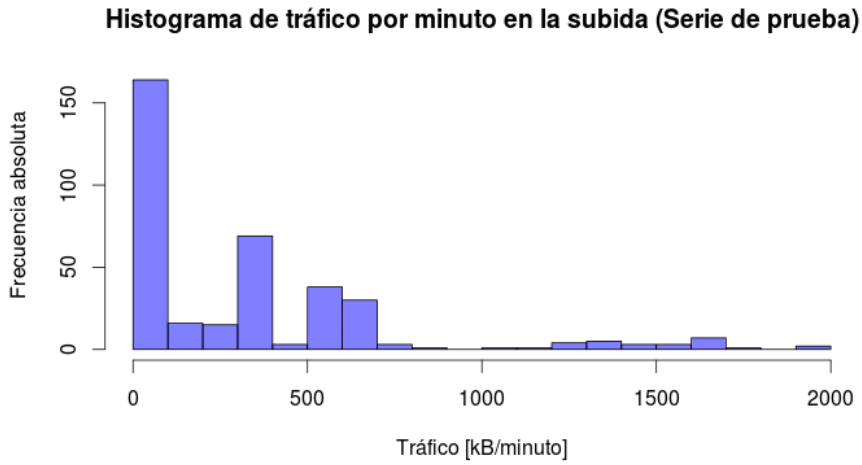

Figura 4.12: Histograma de caracterización de tráfico de entrada al servidor web para la serie de prueba.

Al comparar los histogramas de tráfico de subida en estado de operación normal (figura 4.11) y de tráfico de subida en la serie de prueba (figura 4.12), a primera vista se nota la aparición de flujos de mayor tasa de tráfico (por sobre los 1000 kB/minuto) que corresponden a los minutos en que el sistema se encuentra bajo ataque. Esto indica un buen precedente para la detección mediante Kullback-Leibler, ya que esta diferencia en la distribución permite valores altos calculados para la divergencia durante los ataques, y por lo tanto, una detección exitosa.

4.3.3. Cálculo de Divergencia de Kullback-Leibler

Luego del análisis cualitativo se procedió a calcular la divergencia de Kullback-Leibler para ventanas de 5 minutos en comparación con la caracterización mostrada en la figura 4.11, con lo que se obtuvo lo representado en la figura 4.13.

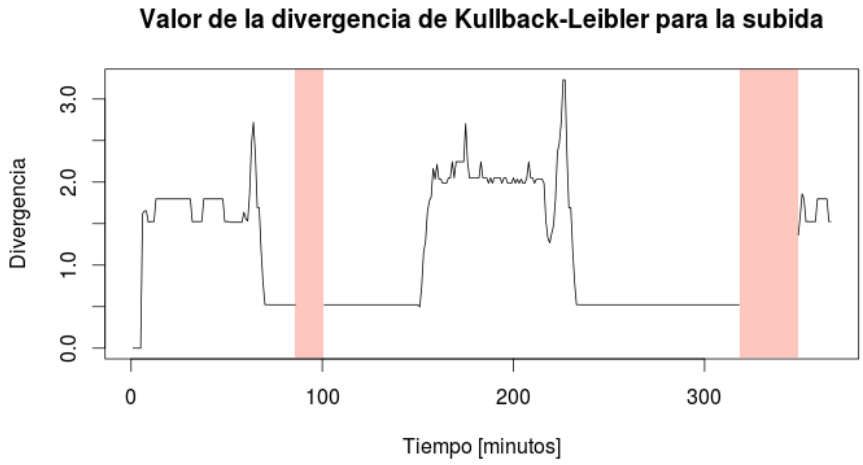

Figura 4.13: Valor de la divergencia de Kullback-Leibler en el tiempo.

Lo que aquí se representa corresponde al valor de la divergencia calculada en cada instante de tiempo, siendo ésta una variable adimensional. Como se puede ver, durante los instantes de ataque (representados con fondo rojo) no aparecen valores graficados. Esto se debe a que el algoritmo de cálculo de la divergencia utilizado (función *KL.empirical* incluida en el paquete"entropy" para R) obtuvo valores demasiado altos y por lo tanto entregó valores "Inf" (equivalente a decir que los valores obtenido tienden a infinito) en toda el área roja. Si bien esto no es lo ideal, comprueba lo intuido en los primeros análisis con respecto a que la gran diferencia de tráfico permitiría la detección exitosa. En efecto, fijando un valor umbral para la divergencia de 3.5, la detección es efectiva dentro de los primeros 5 minutos de comenzado el ataque.

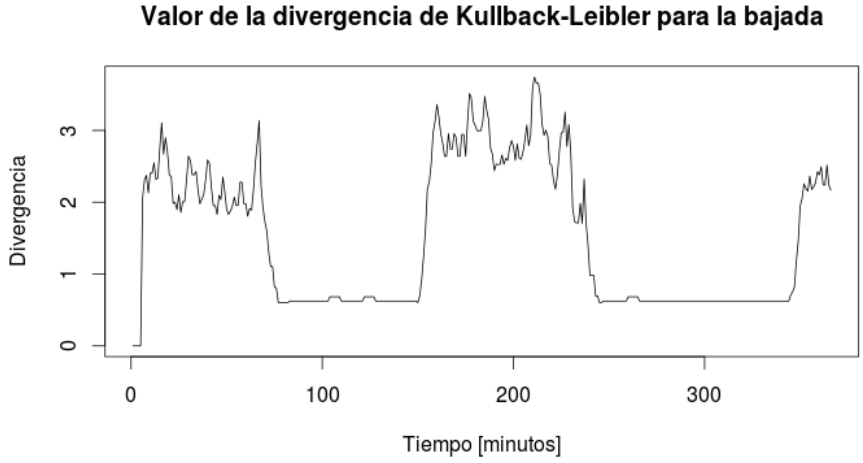

Figura 4.14: Valor de la divergencia de Kullback-Leibler en el tiempo.

Finalmente, a modo de contraste se presenta la divergencia calculada en el caso de utilizar la serie de tiempo de tráfico de bajada (figura 4.14), donde se puede ver que no es posible diferenciar el estado de ataque con respecto al estado de operación normal, y por lo tanto, la detección en este caso no es factible.

Capítulo 5

Conclusiones

5.1. Herramientas de Software

Una de las herramientas fundamentales para el montaje del ambiente de simulación corresponde a GNS3, que demuestra ser fácil de utilizar y permite configurar escenarios de red con alta flexibilidad permitiendo el uso de máquinas virtuales. Estas máquinas pueden contener *software* que se usa en sistemas reales de comunicación, por lo tanto permiten llevar el análisis a escenarios más cercanos a lo que se ve en la práctica.

Por otro lado, *frameworks* como ”Locust” y ”FunkLoad” resultan ser herramientas muy efectivas y simples de usar para ejecutar pruebas de carga a servidores y en particular para este trabajo, permiten generar los niveles de tráfico base y dinámico por separado, dando mayor flexibilidad y realismo en la configuración de los escenarios.

Finalmente, se verifica la utilidad que tiene una *suite* como RStudio para hacer análisis estadísticos de manera fluida, en conjunto con las librerías disponibles de manera libre para R, lo que es clave para avanzar en la obtención de los resultados y generar los reportes necesarios.

5.2. Objetivos Propuestos

En primer lugar, se comprueba la factibilidad de montar un ambiente de simulación representativo de una arquitectura de proveedor de servicios de Internet para realizar estudios e investigaciones relacionadas con redes de comunicaciones y seguridad informática, utilizando hardware disponible en un computador personal y software de código libre. Esto permite entregar todo lo necesario para su configuración en cualquiera de los tres grandes sistemas operativos (Windows, Linux y OSX), incluyendo las máquinas virtuales de la arquitectura y un breve manual para su uso en estudios posteriores.

Además, gracias a la utilización de *frameworks* dedicados y el desarrollo de un modelo

simplificado de comportamiento de usuario, se permite generar el tráfico web necesario que represente distintos niveles de carga para el análisis posterior para la generación de modelos de predicción de tráfico y detección de ataques.

Con respecto a esto último, se verificó la utilidad de aplicar herramientas de inteligencia computacional, clásicamente utilizadas en diversas disciplinas, en sistemas de seguridad de datos. En particular, se presenta el caso de los modelos ARIMA, en donde se demuestra que a pesar de que no se pueden realizar pronósticos precisos para el nivel de tráfico a futuro, sí pueden ser utilizados, por ejemplo, para detectar cuando el nivel de tráfico de entrada está muy fuera del rango pronosticado minutos atrás y por lo tanto, se podría declarar un estado de ataque de denegación de servicio. Esto, por supuesto, debido a las características particulares de éste (cambio repentino de tráfico de subida y nivel promedio hasta tres veces mayor que el tradicional).

Así mismo, y de manera complementaria, se comprobó la factibilidad de implementar un detector basado en la divergencia de Kullback-Leibler para este tipo de ataques, ya que su aplicación para el análisis del tráfico de subida, especialmente, dio excelentes resultados. En particular, durante las simulaciones realizadas permitió la detección de un ataque dentro de los primeros 5 minutos de haber comenzado y sin presentar, a priori, falsas alarmas. Esta situación está sujeta a las condiciones de simulación que fueron posibles de realizar dadas las limitaciones de software y hardware disponibles.

Por último, los resultados obtenidos para la predicción de tráfico y detección mediante la divergencia de Kullback-Leibler presentan, un buen precedente para la aplicación de ambas herramientas dentro de un sistema de detección de intrusos basado en anomalías (*anomaly-based IDS*), permitiendo funcionar de manera complementaria a un sistema tradicional basado en firmas como “Snort”.

5.3. Desarrollo

En general el desarrollo del trabajo presentó complicaciones principalmente durante el montaje y configuración del ambiente de simulación debido a la diversidad de sistemas y aplicaciones utilizadas para emular la arquitectura completa representada. Sin embargo, una vez estabilizado su funcionamiento, el resto del trabajo se completó sin mayores problemas.

Un punto que vale la pena destacar es que, si bien se logran realizar las simulaciones sin mayores contratiempos dentro de un computador personal, las limitantes de hardware disponible no permiten escalar el estudio para sistemas más grandes o eventualmente llevar los niveles de tráfico a valores medidos en la práctica por un proveedor de servicios de Internet, por lo tanto se sugiere usar procesamiento distribuido o incluir más máquinas anfitrionas en el desarrollo del ambiente de simulación.

Vale la pena destacar que a pesar de que se realizaron algunas pruebas para detección de ataques a nivel de aplicación (inyección MySQL), dedicando tiempo a la investigación y utilización de herramientas de análisis de componentes principales y de correspondencia, no se lograron obtener resultados consistentes que justificaran su inclusión en este documento, por lo tanto esta

tarea se añade como sugerencia para trabajo futuro.

5.4. Proyecciones del Trabajo

En cuanto a las proyecciones de este trabajo, como el ambiente de simulación se encuentra configurado y disponible como sistema autocontenido en máquinas virtuales para ”VirtualBox”, existe la posibilidad de trasladarlo a computadores más poderosos permitiendo escalar las simulaciones a niveles de tráfico mayores.

Además está la alternativa de añadir más diversidad en los generadores de tráfico para simular nuevos servicios presentes en Internet como *streaming* de video, servicio de voz sobre IP, transferencia de archivos, etc., en conjunto con el tráfico web ya implementado, de manera de obtener perfiles de tráfico más realistas.

Por otro lado, en cuanto a los tópicos de seguridad informática, existe la posibilidad de probar nuevos ataques utilizando la máquina virtual de Kali Linux disponible en el ambiente de simulación, para contrastar, por ejemplo, el desempeño de la detección mediante la divergencia de Kullback-Leibler para los otros tipos de ataque de denegación de servicio existentes.

Además se propone extender el estudio de las herramientas descartadas por tiempo en este trabajo, como el análisis de componentes principales o de correspondencia, para detectar ataques realizados a nivel de aplicación.

Finalmente, como dentro del ambiente de simulación existen todos los componentes de una red de Internet, éstos se pueden utilizar para realizar otro tipo de estudios relacionados con redes de comunicaciones y por lo tanto las posibilidades se expanden para otras disciplinas ajenas a la seguridad informática.

Bibliografía

[1] Bailey, Michael, Jon Oberheide, Jon Andersen, Zhuoqing Morley Mao, Farnam Jahanian y Jose Nazario: *Automated Classification and Analysis of Internet Malware*. En *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007, Proceedings*, páginas 178–197, 2007. http://dx.doi.org/10.1007/978-3-540-74320-0_10.

[2] Bayer, Ulrich, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Krügel y Engin Kirda: *Scalable, Behavior-Based Malware Clustering*. En *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009. http://www.isoc.org/isoc/conferences/ndss/09/pdf/11.pdf.

[3] Beh, Eric J. y Rosaria Lombardo: *Correspondence Analysis: Theory, Practice and New Strategies (Wiley Series in Probability and Statistics)*. Wiley, 2014, ISBN 1119953243.

[4] Box, George Edward Pelham y Gwilym Jenkins: *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990, ISBN 0816211043.

[5] Chatfield, C.: *Time-Series Forecasting*. CRC Press, 2000, ISBN 9781420036206.

[6] Chatfield, Chris: *The analysis of time series: an introduction*. CRC Press, Florida, US, 6th edición, 2004.

[7] Cover, T.M. y J.A. Thomas: *Elements of information theory*. Wiley-Interscience, 2006.

[8] DeFino, Steven y Larry Greenblatt: *Official Certified Ethical Hacker Review Guide: For Version 7.1*. Course Technology Press, Boston, MA, United States, 1st edición, 2012, ISBN 1133716288, 9781133716280.

[9] George Athanasopoulos, Rob J Hyndman with contributions from, Slava Razbash, Drew Schmidt, Zhenyu Zhou nd Yousaf Khan y Christoph Bergmeir: *forecast: Forecasting functions for time series and linear models*, 2013. http://CRAN.R-project.org/package=forecast, R package version 4.8.

[10] Hausser, Jean y Korbinian Strimmer: *entropy: Estimation of Entropy, Mutual Information and Related Quantities*, 2013. http://CRAN.R-project.org/package=entropy, R package version 1.2.0.

[11] Jolliffe, I. T.: *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002, ISBN 9780387954424.

[12] Kullback, Solomon: *Information Theory and Statistics*. Wiley, New York, 1959.

[13] Mohammad Masoud Javidi, Mohammad Hassan Nattaj: *A New and Quick Method to Detect DoS Attacks by Neural Networks*. Journal of mathematics and computer Science, 6:85–96, 2013.

[14] Rieck, Konrad: *Computer Security and Machine Learning: Worst Enemies or Best Friends?* SysSec Workshop, 0:107–110, 2011.

[15] Robertson, William K., Federico Maggi, Christopher Kruegel y Giovanni Vigna: *Effective Anomaly Detection with Scarce Training Data*. En *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, 2010. http://www.isoc.org/isoc/conferences/ndss/10/pdf/07.pdf.

[16] Ryan, Jeffrey A. y Joshua M. Ulrich: *xts: eXtensible Time Series*, 2014. http://CRAN.R-project.org/package=xts, R package version 0.9-7.

[17] See, Garret: *Quantmod Add-on*, 2014. http://r-forge.r-project.org/R/?group_id=1113, R package version 1.6.10.

[18] Song, Yingbo, Angelos D. Keromytis y Salvatore J. Stolfo: *Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic*. En *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009. http://www.isoc.org/isoc/conferences/ndss/09/pdf/07.pdf.

[19] Stallings, William: *Network Security Essentials - Applications and Standards (4. ed., internat. ed.)*. Pearson Education, 2010, ISBN 978-0-13-610805-4.

[20] Stoffer, David: *astsa: Applied Statistical Time Series Analysis*, 2012. http://CRAN.R-project.org/package=astsa, R package version 1.1.

[21] Stokely, Murray: *HistogramTools: Utility Functions for R Histograms*, 2013. http://CRAN.R-project.org/package=HistogramTools, R package version 0.3.

[22] Stuart, Alan, Keith Ord y Steven Arnold: *Kendall's Advanced Theory of Statistics, Classical Inference and the Linear Model (Volume 2A)*. Wiley, 2010, ISBN 0470689242.

[23] Zeileis, Achim y Torsten Hothorn: *Diagnostic Checking in Regression Relationships*. R News, 2(3):7–10, 2002. http://CRAN.R-project.org/doc/Rnews/.

Lista de acrónimos

ACF: Autocorrelation Function.

AIC: Akaike Information Criterion.

AR: Autoregresive.

ARIMA: Autoregresive Integrated Moving Average.

ARMA: Autoregresive Moving Average.

CIA: Confidentiality, Integrity and Accountability.

CSV: Comma Separated Values.

DHCP: Dynamic Host Configuration Protocol.

DoS: Denial of Service.

GNS3: Graphical Network Simulator 3.

GPL: General Public License.

HTML: HyperText Markup Language.

IDE: Integrated Development Environment.

IDS: Intrusion Detection System.

IP: Internet Protocol

ISP: Internet Service Provider.

ITU: International Telecommunication Union.

MA: Moving Average.

NIST: National Institute of Standards and Technology.

OSI: Open System Interconnection.

PACF: Partial Autocorrelation Function.

SARIMA: Seasonal Autoregresive Integrated Moving Average.

SQL: Structured Query Language.

SYN: Synchronize packet.

TCP: Transmission Control Protocol.

UDP: User Datagram Protocol

XSS: Cross-site Scripting.

Apéndice A

Scripts en R

A.1. Estadística Descriptiva y Modelo de Predicción de Tráfico Web

```
# Script de analisis de datos para modelo de prediccion.

# Importar librerias necesarias.
library(xts)
library(stats)
library(forecast)
library(entropy)
library(lattice)
library(lmtest)
library(qmao)


dump_router3 <- read.csv("/media/host1/Dumps/Carga/router3.csv",
    stringsAsFactors = FALSE)

############################## Preprocesamiento ##############################

n=nrow(dump_router3)
dump_router3 <- dump_router3[-c(n-2, n-1, n),]

# Extraer de las columnas a utilizar.
TS_router3 <- dump_router3[,c("tr", "sa", "sp", "da", "dp", "pr", "ipkt", "
    ibyt")]
# LIBERAR MEMORIA RAM
rm(dump_router3)

# Convertir sa, sp, da, dp a etiquetas (factors).

TS_router3$sa <- as.factor(TS_router3$sa)
TS_router3$sp <- as.factor(TS_router3$sp)
TS_router3$da <- as.factor(TS_router3$da)
```

```
TS_router3$dp <- as.factor(TS_router3$sp)
TS_router3$pr <- as.factor(TS_router3$pr)

# Convertir a timestamp strings de columna "tr".

TS_router3$tr <- as.POSIXct(strptime(TS_router3$tr,"%Y-%m-%d %H:%M:%OS"))

############################ Series de Tiempo ##############################

# Crear series de tiempo con datos de bytes enviados agregados por tiempo.
t = 1 # Periodo de agregacion en minutos.

# Direcciones IP.
serv_addr = "10.0.30.10"
clie_addr = "10.0.10.10"

# Router 3
TS_r3_http_in <- TS_router3[TS_router3$da == serv_addr, ]
TS_r3_http_in <- aggregate(ibyt ~ tr, data = TS_r3_http_in, FUN = sum)

xts.r3_http_in <- xts(TS_r3_http_in$ibyt, TS_r3_http_in$tr)
TS_r3_http_in_by_min <- period.apply(xts.r3_http_in, endpoints(xts.
    r3_http_in, "minutes", k=t), sum)
TS_r3_http_in_by_min <- align.time(TS_r3_http_in_by_min, 60)

TS_r3_http_out <- TS_router3[TS_router3$da == clie_addr, ]
TS_r3_http_out <- aggregate(ibyt ~ tr, data = TS_r3_http_out, FUN = sum)

xts.r3_http_out <- xts(TS_r3_http_out$ibyt, TS_r3_http_out$tr)
TS_r3_http_out_by_min <- period.apply(xts.r3_http_out, endpoints(xts.
    r3_http_out, "minutes", k=t), sum)
TS_r3_http_out_by_min <- align.time(TS_r3_http_out_by_min, 60)

#################### INGRESO DE MUESTRA DE PRUEBA ########################
# Ingresar muestra de prueba.
test_sample <- read.csv("/media/host1/Dumps/DoS_Test1/router3.csv",
    stringsAsFactors = FALSE)
n= nrow(test_sample)
test_sample <- test_sample[-c(n-2, n-1, n),]

TS_test <- test_sample[,c("tr", "sa", "sp", "da", "dp", "pr", "ipkt", "ibyt
    ")]
TS_test$sa <- as.factor(TS_test$sa)
TS_test$sp <- as.factor(TS_test$sp)
TS_test$da <- as.factor(TS_test$da)
TS_test$dp <- as.factor(TS_test$sp)
TS_test$pr <- as.factor(TS_test$pr)
TS_test$tr <- as.POSIXct(strptime(TS_test$tr,"%Y-%m-%d %H:%M:%OS"))

TS_r3_http_in_test <- TS_test[TS_test$da == serv_addr, ]
TS_r3_http_in_test <- aggregate(ibyt ~ tr, data = TS_r3_http_in_test, FUN =
    sum)
```

```
xts.r3_http_in_test <- xts(TS_r3_http_in_test$ibyt, TS_r3_http_in_test$tr)
TS_r3_http_in_by_min_test <- period.apply(xts.r3_http_in_test, endpoints(xts
    .r3_http_in_test, "minutes", k=t), sum)
TS_r3_http_in_by_min_test <- align.time(TS_r3_http_in_by_min_test, 60)
TS_r3_http_in_by_min_test <- MakeStrictlyRegular(TS_r3_http_in_by_min_test,
    by = "min", maxgap = 1)

TS_r3_http_out_test <- TS_test[TS_test$sa == serv_addr, ]
TS_r3_http_out_test <- aggregate(ibyt ~ tr, data = TS_r3_http_out_test, FUN
    = sum)

xts.r3_http_out_test <- xts(TS_r3_http_out_test$ibyt, TS_r3_http_out_test$tr
    )
TS_r3_http_out_by_min_test <- period.apply(xts.r3_http_out_test, endpoints(
    xts.r3_http_out_test, "minutes", k=t), sum)
TS_r3_http_out_by_min_test <- align.time(TS_r3_http_out_by_min_test, 60)
TS_r3_http_out_by_min_test <- MakeStrictlyRegular(TS_r3_http_out_by_min_test
    , by = "min", maxgap = 5)


####################### Estadistica descriptiva #########################

# Graficar series de tiempo.
par(mfrow = c(1,1))

# Router 3.
# Series de tiempo estado normal.
plot(TS_r3_http_in_by_min/1000, main = "Trafico de entrada al servidor web a
     traves del router No.3", type = "l", xlab = "Tiempo [minutos]", ylab =
    "Trafico [kB/minuto]")
plot(TS_r3_http_out_by_min/1000, main = "Trafico de salida del servidor web
    a traves del router No.3", type = "l", xlab = "Tiempo [minutos]", ylab =
     "Trafico [kB/minuto]")

# Series de tiempo de prueba. Inlcuye normal y ataques DoS.
plot(TS_r3_http_in_by_min_test/1000, main = "Trafico de entrada al servidor
    web a traves del router No.3", type = "l", xlab = "Tiempo [minutos]",
    ylab = "Trafico [kB/minuto]")
plot(TS_r3_http_out_by_min_test/1000, main = "Trafico de salida del servidor
     web a traves del router No.3", type = "l", xlab = "Tiempo [minutos]",
    ylab = "Trafico [kB/minuto]")


# Dividir y agregar series para estado de operacion normal y en falla.
# Tiempos de inicio y fin de tramos de operacion normal.

T_normal <- c("2015-09-15 20:50:00/2015-09-16 12:00:00")
T_test <- c("2015-10-13 18:15:00/2015-10-14 00:20:00")

# Agregar series de tiempo segun estado de operacion. [kB/min]
ts_normal_r3 <- data.frame(N = sequence(c(nrow(TS_r3_http_in_by_min[T_normal
    ,]))), up = TS_r3_http_in_by_min[T_normal,]/1000, down =
    TS_r3_http_out_by_min[T_normal,]/1000)
```

```
ts_test_r3 <- data.frame(N = sequence(c(nrow(TS_r3_http_in_by_min_test[
    T_test,]))), up = TS_r3_http_in_by_min_test[T_test,]/1000, down =
    TS_r3_http_out_by_min_test[T_test,]/1000)


# Resumen estadistico.
stat_summ_normal <- numSummary(ts_normal_r3[,c("up","down")], statistics=c("
    mean", "quantiles", "sd", "cv", "kurtosis"),
                               quantiles=c(0,.25,.5,.75,1), type="2")

stat_summ_test <- numSummary(ts_test_r3[,c("up","down")], statistics=c("mean
    ", "quantiles", "sd", "cv", "kurtosis"),
                             quantiles=c(0,.25,.5,.75,1), type="2")

# Histogramas.

# Operacion normal vs bajo ataque.
# Subida
# Router 3
hist_normal_up_r3 <- hist(ts_normal_r3$up, 50, col=rgb(0,0,1,0.5), xlab = "
    Tasa de trafico [kB/minuto]", ylab = "Frecuencia",
                          main = "Histograma de trafico de entrada al
                              servidor a traves del router No.3")

hist_dos_up_r3 <- hist(ts_test_r3$up, 50, col=rgb(1,0,0,0.5))

# Bajada
# Router 3
hist_normal_down_r3 <- hist(ts_normal_r3$down, 50, col=rgb(0,0,1,0.5), xlab
    = "Tasa de trafico [kB/minuto]", ylab = "Frecuencia",
                            main = "Histograma de trafico de salida al
                                servidor a traves del router No.3")

hist_dos_down_r3 <- hist(ts_test_r3$down, 50, col=rgb(1,0,0,0.5))

## Aca insertar estadistica basica de operacion en DoS.


# Boxplots Normal v/s DoS.
# Subida
boxplot(ts_normal_r3$up, names = c("Normal"), ylab = "Trafico [kB/minuto]",
        main = "Boxplots de trafico de subida a traves del router No. 3")

# Bajada
boxplot(ts_normal_r3$down, names = c("Normal"), ylab = "Trafico [kB/minuto
    ]",
        main = "Boxplots de trafico de bajada a traves del router No. 3")

# Comparacion
boxplot(ts_normal_r3$up, ts_normal_r3$down, names = c("Subida", "Bajada"),
    ylab = "Trafico [kB/minuto]",
        main = "Boxplots de trafico traves del router No. 3")
```

```
######################### Modelos de Prediccion ##########################

# Extractos de serie de tiempo para entrenamiento y validacion.
factor = 0.4
m = round(factor*nrow(ts_normal_r3))
ts_train <- ts_normal_r3[ts_normal_r3$N <= m,]
ts_valid <- ts_normal_r3[ts_normal_r3$N >= m+1,]

# SUBIDAen
# Para la prediccion se seleccionaran los datos del router 3.
Modelo_pred_up = arima(ts_train$up)
# Revision de modelo y coeficientes significativos o no
Modelo_pred_up
coeftest(Modelo_pred_up)
# Guardar residuos y ajuste de modelo seleccionado
ts_train$Residuos_Up <- c(Modelo_pred_up$res)
ts_train$Ajuste_Up <- ts_train$up - ts_train$Residuos_Up
# head(Base1)
tsdiag(Modelo_pred_up)
title("Analisis modelo de regresion simple para la Subida", line = 3.5, font
    .main = 1, cex.main = 0.7)

# Generacion de modelos.

# Analisis ARIMA
plot(ts_train$up, main = "Trafico de subida y Ajuste", bty = "n", lwd = 2,
    type = "l")
lines(ts_train$Ajuste_Up, col = "blue", bty = "n", lwd = 2, type = "l")
# ACF, PACF
par(mfrow = c(1,2))
Lag = 20
acf(ts_train$Residuos_Up, col = 2, lwd = 2, main = "", bty = "n" , ylim = c
    (-1,1), lag.max = Lag)
title("ACF")
pacf(ts_train$Residuos_Up, col = 2, lwd = 2, main = "", bty = "n" , ylim = c
    (-1,1), lag.max = Lag, xlim = c(0,Lag))
title("PACF")

# ACF, PACF
par(mfrow = c(1,2))
Lag = 20
acf(ts_train$up, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1),
    lag.max = Lag)
title("ACF")
pacf(ts_train$up, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1),
    lag.max = Lag, xlim = c(0,Lag))
title("PACF")

# Analisis diferenciacion
X1 = diff(ts_train$up)
X2 = diff(diff(ts_train$up), lag = 1)
#
# Graficas de ACF y PACF de Subida diferenciada una y dos veces
```

```
par(mfrow = c(1,2))
Lag = 20
acf(X1, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1), lag.max =
    Lag)
title("ACF diff X1")
pacf(X1, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1), lag.max =
    Lag, xlim = c(0,Lag))
title("PACF diff X1")

par(mfrow = c(1,2))
Lag = 20
acf(X2, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1), lag.max =
    Lag)
title("ACF diff X2")
pacf(X2, col = 2, lwd = 2, main = "", bty = "n" , ylim = c(-1,1), lag.max =
    Lag, xlim = c(0,Lag))
title("PACF diff X2")

MaxOrdenR = c(3,1,3)
AICReg = 1e10
AICiterR = c()
iterReg = 0
for (p in 0:MaxOrdenR[1]) for(d in 0:MaxOrdenR[2]) for(q in 0:MaxOrdenR[3])
{
  print("=========================================")
  fitReg = arima(ts_train$up, order = c(p,d,q), include.mean = F)
  print(c('Modelo','p=',p,'q=',q))
  print(fitReg)
  print("-----------------------------------------")
  iterReg = iterReg + 1
  print(c('AIC de modelo ARMA','iteracion # =',iterReg))
  AICfitReg = AIC(fitReg)
  print(AICfitReg)
  AICiterR[iterReg] = AICfitReg
  if (AICfitReg < AICReg)
  {
    AICReg <- AICfitReg
    Modelo.fitReg <- fitReg
    ModelReg.orden <- c(p,d,q)
  }
}

list(AICReg, Modelo.fitReg, ModelReg.orden)
AICiterR
min(AICiterR)

# Prediccion con modelo ARIMA
Pred_up = forecast(Modelo.fitReg, h = 60)
# Grafico prediccion
par(mfrow = c(1,1))
plot(Pred_up, include = 100, main = "Pronostico con modelo ARIMA(2,1,0)",
     ylab = "Trafico [kB/minuto]", xlab = "Tiempo [minutos]")
```

```
##########################################################
```

A.2. Detección utilizando la divergencia de Kullback-Leibler

```
# Script de analisis de datos para modelo de deteccion.
# Importar librerias necesarias.
library(xts)
library(stats)
library(forecast)
library(entropy)
library(lattice)
library(lmtest)
library(astsa)
library(HistogramTools)

#################### CARACTERIZACION DE ESTADO NORMAL #################

# Histogramas
hist_normal_up <- hist(ts_normal_r3$up, seq(0,2000,100), col=rgb(0,0,1,0.5),
                    main = 'Histograma de trafico por minuto en la subida',
                        xlab = 'Trafico [kB/minuto]',
                    ylab = 'Frecuencia')

hist_normal_down <- hist(ts_normal_r3$down, seq(0,14000,500), col=rgb
    (0,0,1,0.5),
                      main = 'Histograma de trafico por minuto en la bajada
                          ', xlab = 'Trafico [kB/minuto]',
                      ylab = 'Frecuencia')

hist_test_up <- hist(ts_test_r3$up, seq(0,2000,100), col=rgb(0,0,1,0.5),
                       main = 'Histograma de trafico por minuto en la subida
                            (Serie de prueba)', xlab = 'Trafico [kB/minuto
                           ]',
                       ylab = 'Frecuencia')

hist_test_down <- hist(ts_test_r3$down, seq(0,14000,500), col=rgb(0,0,1,0.5)
    ,
                         main = 'Histograma de trafico por minuto en la
                             bajada (Serie de prueba)', xlab = 'Trafico [kB/
                             minuto]',
                         ylab = 'Frecuencia')

########################### DETECCION ############################

# Subida
hist(ts_test_r3$up)
tamano_ventana = 5 # minutos
kl_subida <- vector(length = NROW(ts_test_r3$up))
```

```
estado_subida <- vector(length = NROW(ts_test_r3$up)) # marcador de estado
    0: normal, 1: DoS.
# Recorrer serie
for(i in 1:(NROW(ts_test_r3$up) - tamano_ventana))
  {
    hist_ventana <- hist(ts_test_r3$up[i:(i+tamano_ventana)],seq(0,2000,100)
        , plot = FALSE)
    kl_subida[i+tamano_ventana] <- KL.empirical(unlist(hist_ventana["counts
        "]), unlist(hist_normal_up["counts"]))
  }
plot(kl_subida, type = "l", main = "Valor de la divergencia de Kullback-
    Leibler para la subida",
     xlab = "Tiempo [minutos]", ylab = "Divergencia")


# Bajada
hist(ts_test_r3$up)
kl_bajada <- vector(length = NROW(ts_test_r3$down))
estado_bajada <- vector(length = NROW(ts_test_r3$down)) # marcador de estado
     0: normal, 1: DoS.
# Recorrer serie
for(i in 1:(NROW(ts_test_r3$down) - tamano_ventana))
{
  hist_ventana <- hist(ts_test_r3$down[i:(i+tamano_ventana)], seq
      (0,14000,500), plot = FALSE)
  kl_bajada[i+tamano_ventana] <- KL.empirical(unlist(hist_ventana["counts"])
      ,unlist(hist_normal_down["counts"]))
}
kl_bajada
plot(kl_bajada, type = "l", main = "Valor de la divergencia de Kullback-
    Leibler para la bajada",
     xlab = "Tiempo [minutos]", ylab = "Divergencia")
```

Apéndice B

Scripts de Modelo de Comportamiento de Usuario

B.1. Script para Locust

```
import resource
import random
from locust import HttpLocust, TaskSet, task
from pyquery import PyQuery

class BrowseDocumentation(TaskSet):
    def on_start(self):
        # assume all users arrive at the index page
        self.index_page()
        self.urls_on_current_page = self.toc_urls

    @task(10)
    def index_page(self):
        r = self.client.get("/")
        pq = PyQuery(r.content)
        link_elements = pq('a')
        self.toc_urls = [
            l.attrib["href"] for l in link_elements
        ]

    @task(50)
    def load_page(self, url=None):
        url = random.choice(self.toc_urls)
        r = self.client.get(url)
        pq = PyQuery(r.content)
        link_elements = pq('a')
        self.urls_on_current_page = [
            l.attrib["href"] for l in link_elements
        ]
```

```
    @task(30)
    def load_sub_page(self):
        url = random.choice(self.urls_on_current_page)
        r = self.client.get(url)

class AwesomeUser(HttpLocust):
    task_set = BrowseDocumentation
    host = "http://10.0.30.10"
    min_wait = 0  * 1000
    max_wait = 120 * 1000
```

B.2. Script para FunkLoad

```
import random
import urlparse

from funkload.FunkLoadTestCase import FunkLoadTestCase

class Dinamico(FunkLoadTestCase):
        def setUp(self):
                self.server_url = self.conf_get('main', 'url')
        def test_Dinamico(self):
                base_url = self.server_url
                while True:
                        self.get(base_url, description='Get url')
                        body = self.getBody()
                        links = self.listHref()
                        self.sleep()
                        base_url = urlparse.urljoin(base_url,random.choice(
                            links))
                        print(base_url)
```

B.3. Archivo de Configuración para FunkLoad

```
[main]
title = Demo
description = Simulacion
url = http://10.0.30.10

[test_Dinamico]
description = Simulacion Carga Dinamica

# Configuracion de los ciclos de carga
```

```
[bench]

# Definir cantidad de usarios por ciclo.
cycles = 400:800:400:800:400

# Duracion de cada ciclo.
duration = 3600
startup_delay = 0.5
sleep_time = 0
cycle_time = 3600
log_to =
log_path = simple-bench.log
result_path = simple-bench.xml

#Tiempo de espera de cada usuario.
sleep_time_min = 0
sleep_time_max = 120
```

Apéndice C

Guía de Uso del Ambiente de Simulación

A continuación se detallan las instrucciones para el montaje y operación del ambiente de simulación desarrollado. Vale la pena destacar que a pesar de que se intentó compactar todo lo necesario dentro de una máquina virtual dedicada para el simulador, las limitaciones del software de virtualización para la capacidad de procesamiento impidieron un funcionamiento adecuado de ésta, por lo tanto se debe realizar un breve procedimiento de instalación de software e importación de máquinas virtuales para montar el ambiente de simulación.

C.1. Instalación de Programas Requeridos

Para el montaje del ambiente de simulación es necesario instalar Virtualbox y GNS3 en el computador principal. Ambos programas tienen versiones disponibles para los tres grandes sistemas operativos (Windows, Linux y OSX) y cuentan con asistentes de instalación tradicionales, por lo tanto no deberían presentar mayor complicación.

Los instaladores pueden ser descargados desde la siguientes direcciones:

- Virtualbox: http://www.virtualbox.org
- GNS3: http://www.gns3.com

C.2. Importación de Máquinas Virtuales

Una vez instalado Virtualbox, se debe proceder a importar las máquinas virtuales que conforman la arquitectura simplificada de ISP implementada. Para esto se deben seguir los siguientes pasos:

1. Abrir Virtualbox.

2. En la barra de menús ir a: *File* → *Import appliance*. Con esto se abrirá el asistente de importación de máquinas virtuales de Virtualbox.

3. Seleccionar archivo ”VMs_Simulador.ova” entregado que contiene las máquinas virtuales del ambiente de simulación.

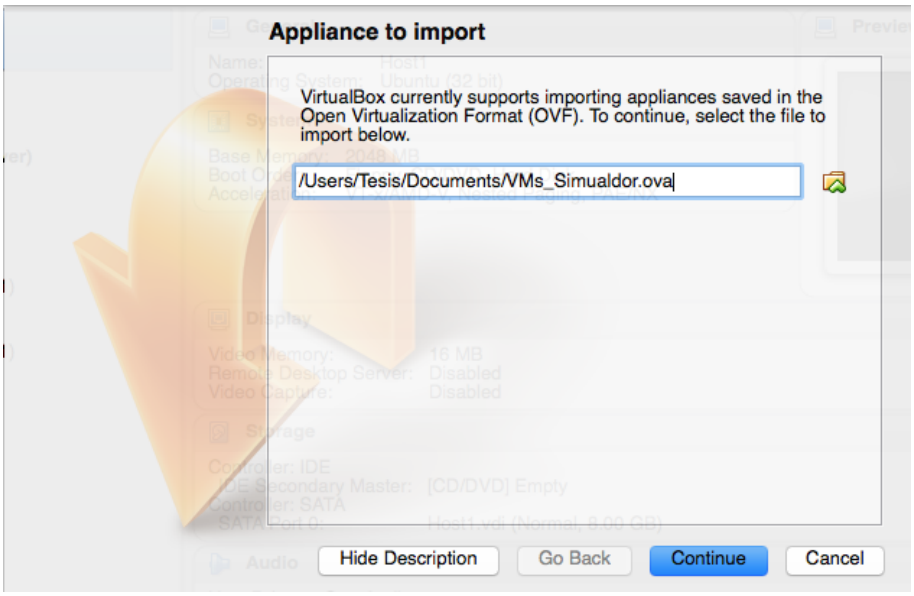

Figura C.1: Asistente de importación de máquinas virtuales de Virtualbox.

4. Luego presionar *Continue* y posteriormente *Import*. Finalmente luego de unos minutos se tendrán todas las máquinas virtuales listas para ser utilizadas en GNS3 como se muestra en la figura C.2.

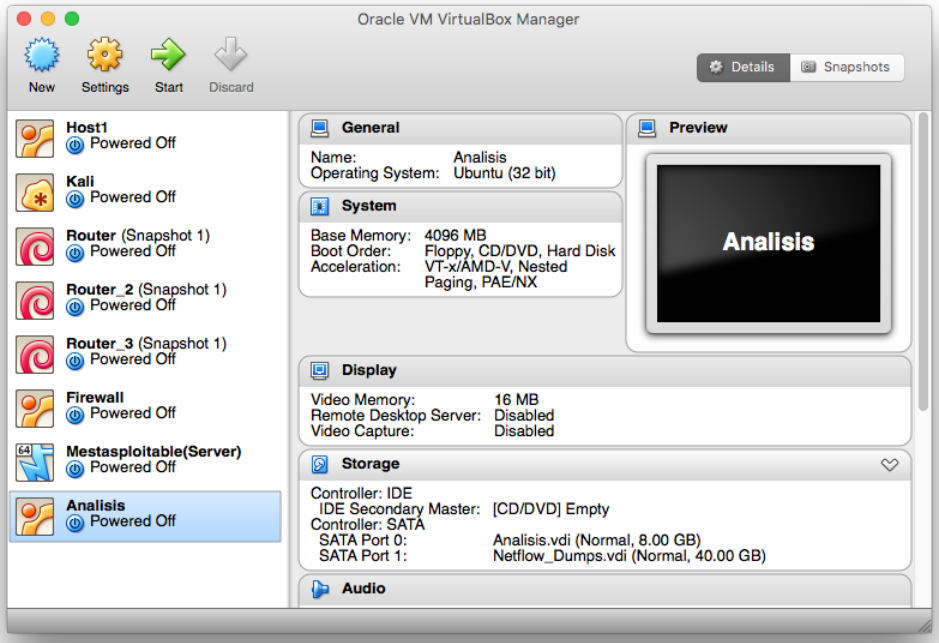

Figura C.2: Virtualbox con las máquinas virtuales del simulador importadas.

Con esto queda todo listo para utilizar el proyecto desarrollado en GNS3, y por lo tanto ejecutar las simulaciones.

C.3. Proyecto en GNS3

Dentro de los archivos entregados, se encuentra la carpeta del proyecto en GNS3 con el cual se implementó la arquitectura del ambiente de simulación, utilizando las máquinas virtuales importadas anteriormente. En la figura C.3 se puede ver el contenido del proyecto, en donde el archivo más importante corresponde a ”Simulador_ISP.gns3”.

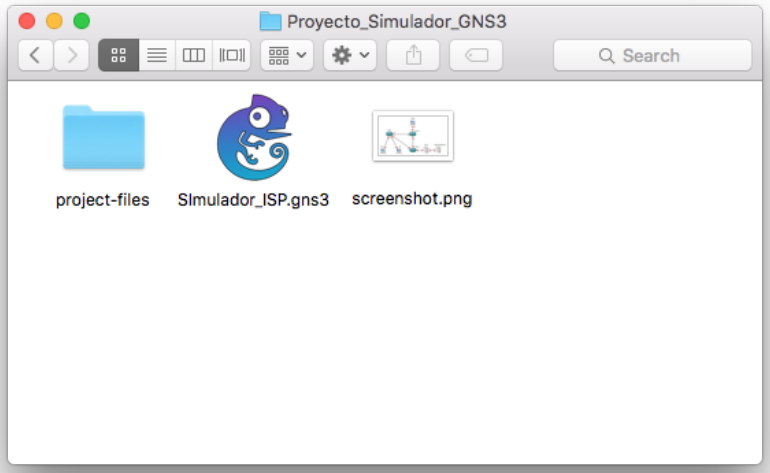

Figura C.3: Contenido de la carpeta del proyecto en GNS3 asociado al simulador.

Para su utilización basta con transferir la carpeta al computador principal donde se estime conveniente y estará todo listo para ejecutar una simulación.

C.4. Ejecución de una Simulación

C.4.1. Resumen de Procedimiento

Antes de comenzar una simulación es necesario verificar que todos los sistemas están correctamente configurados y que la red emulada funcionará sin problemas. Luego de la verificación inicial se procede a ejecutar la simulación según la siguiente lista de pasos:

1. Iniciar máquinas virtuales.
2. Iniciar la captura de registros de Netflow en la máquina dedicada (Análisis).
3. Iniciar generador de tráfico con cantidad de usuarios estática (carga base) en cliente (Host1).
4. Iniciar generador de tráfico con cantidad de usuarios variable (carga dinámica) en cliente (Host1).
5. Ejecutar ataques desde máquina atacante (Kali).
6. Terminar generadores de tráfico y recolectores de información.

7. Compilar registros en archivos CSV en la máquina sonda (Análisis)

8. Terminar la sesión en GNS3 presionando el botón ”Stop” ubicado en la barra de herramientas superior.

C.4.2. Detalles del Procedimiento para Ejecución de una Simulación

Inicio de Máquinas Virtuales

Para iniciar una simulación se debe ejecutar el archivo ”Simulador_ISP.gns3”, con lo que se mostrará en pantalla el ambiente de simulación desarrollado como se muestra en la figura C.4.

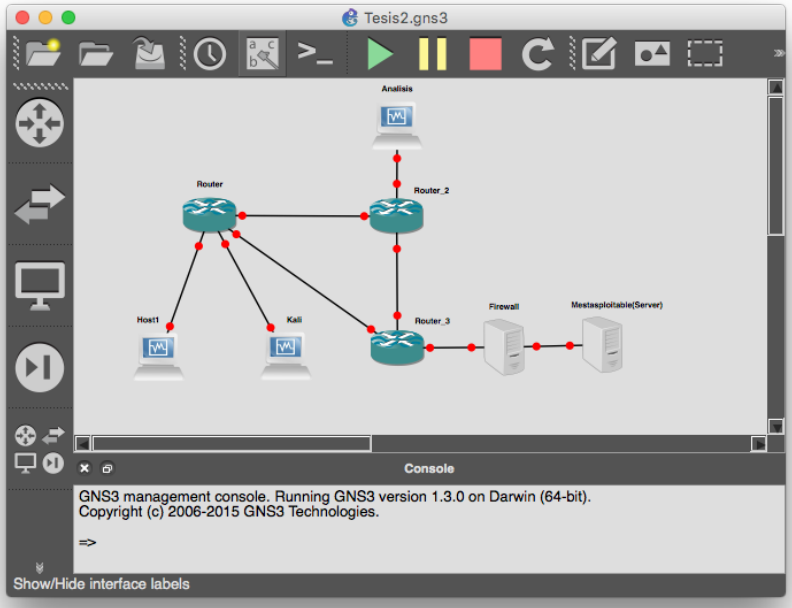

Figura C.4: Vista de GNS3 para el proyecto del ambiente de simulación implementado.

Luego de esto se debe presionar el botón *Play* ubicado en la barra superior de herramientas de GNS3 y con esto se iniciarán todas las máquinas virtuales del sistema.

Captura de Registros en Sonda vía Nfcap

Para capturar en la sonda los registros envíados por cada uno de los enrutadores, es necesario iniciar tres instancias de recolección utilizando ”nfcap”. Para esto se deben seguir los siguientes pasos:

1. Iniciar sesión en máquina asociada a la sonda (Análisis).

2. Abrir tres ventanas de terminal y en cada una de ellas ejecutar el siguiente comando:

```
$ nfcapd -w -T all -p XXXX -I 'routerN'
  -l /home/host1/Analisis/Netflow/routerN -s 2
```

   en donde XXXX corresponde a los puertos 2055, 2056 y 2057 para recibir la información de los enrutadores 1, 2 y 3 respectivamente, y 'routerN' corresponde a un nombre seleccionado para identificarlos.

3. Luego, una vez terminada la simulación, se debe terminar el proceso de recolección en las tres instancias iniciadas, presionando *Ctrl+C* o simplemente cerrando las ventanas asociadas. Los registros quedarán almacenados en bruto en la ubicación específicada en el comando anterior.

Generación de Tráfico con Cantidad de Usuarios Estática Utilizando Locust

Para generar la carga de tráfico base es necesario ejecutar Locust con el *script* previamente programado que implementa el modelo de usuario propuesto. Esto se logra siguiendo los siguientes pasos:

1. Iniciar sesión en la máquina asociada al cliente tradicional.

2. Abrir una ventana de terminal, ir a la carpeta en donde se encuentra almacenado el *script* de comportamiento de usuario para Locust y ejecutar el siguiente comando:

```
$ locust -f test\_Base.py
```

   donde "test_Base.py.[es] el nombre asignado al *script* desarrollado e incluido como anexo a este documento.

3. Luego, se debe acceder a través del navegador web a la dirección **http://127.0.0.1:8089/** para acceder ala interfaz gráfica de Locust.

4. Seleccionar cantidad de usuarios a simular y tasa de inicio de las instancias (*hatch rate*), como se muestra en la figura C.5.

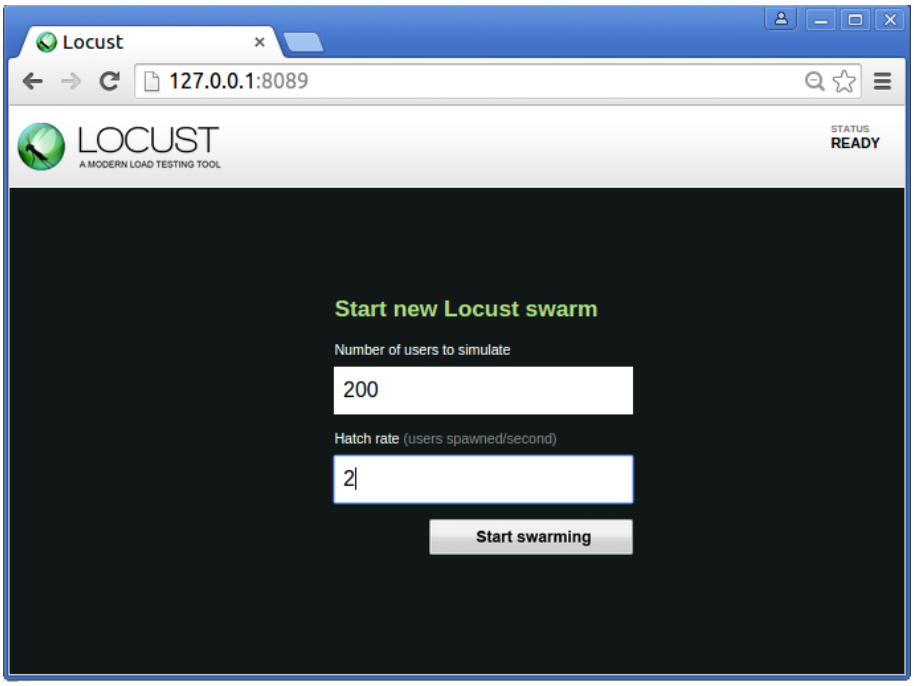

Figura C.5: Interfaz gráfica de Locust para configuración y manejo del generador de tráfico basal.

5. Presionar ”Start Swarming”.

6. Una vez terminada la simulación se puede presionar el botón ”Stop”de la interfaz de Locust para terminar las instancias iniciadas.

Generación de Tráfico con Cantidad de Usuarios Dinámica Utilizando FunkLoad

Para generar la carga de tráfico dinámica es necesario ejecutar Funkload con el *script* previamente programado que implementa el modelo de usuario propuesto. Esto se logra siguiendo los siguientes pasos:

1. Iniciar sesión en la máquina asociada al cliente tradicional.

2. Abrir una ventana de terminal, ir a la carpeta en donde se encuentra almacenado el *script* de comportamiento de usuario para Funkload y ejecutar el siguiente comando:

```
$ fl-run-bench test_Dinamico.py Dinamico.test_dinamico
```

   donde ”test_Dinamico.py” es el nombre asignado al *script* desarrollado e incluído como anexo a este documento.

Ejecución de Ataques

Para cada ejecución de un ataque ”SYN-flood”fue necesario seguir la secuencia de pasos descrita a continuación utilizando ”Metasploitçomo herramienta principal.

1. Iniciar sesión en la máquina atacante (Kali).
2. Abrir ”Metasploit”dentro de una ventana de terminal ejecutando la siguiente instrucción.

```
$ msfconsole
```

3. Iniciar programa auxiliar para ataque SYN-flood.

```
msf > use auxiliary/dos/tcp/synflood
```

4. Configurar ataque, asignando dirección IP y puerto de destino. En particular para este trabajo, la dirección del servidor web es 10.0.30.10:80.

```
msf > set rhost 10.0.30.10
```

5. Ejecutar ataque.

```
msf > exploit
```

6. Finalmente, luego de esperar el tiempo necesario para la recolección de datos, se debe terminar el programa para volver al estado normal de operación.

```
msf > exit
```

Almacenamiento Definitivo en Formato CSV

Finalmente para el proceso de almacenamiento de los registros, se utilizaron archivos de valores separados por coma (.csv) extraídos desde los archivos en bruto generados por el capturador de flujos IP. La ventaja de utilizar archivos CSV es que son fácilmente manejados e importados a RStudio. Para esto se utilizó la herramienta ”nfdump”disponible a través del terminal de la máquina virtual asociada a la sonda, siguiendo los siguientes pasos:

1. Iniciar sesión en máquina asociada a la sonda.
2. Abrir ventana de terminal y ejecutar la siguiente secuencia para compilar la información correspondiente de cada enrutador .csv independientes:

```
$ nfdump -R /home/host1/Netflow/routerN -o csv >>
  /media/host1/Dumps/routerN.csv
```

donde ”routerN” puede ser nuevamente un identificador seleccionado para cada enrutador.

C.5. Utilización de Scripts en R

En la máquina virtual de análisis de datos se encuentran incluidos como ejemplo los *scripts* utilizados en el trabajo para la obtención de los resultados. Éstos corresponden a dos archivos en R, dentro de la carpeta ”Scripts R” ubicada en el escritorio, que completan las tareas de manera automática.

Los resultados que generan los *scripts* son:

- **”Modelo_prediccion.r”**
  - Gráficos de serie de tiempo.
  - Resumen de estadística básica.
  - Histogramas, *Box-plots*.
  - Gráficos de ACF y PACF.
  - Detalles del modelo ARIMA de predicción de tráfico.
- **”Deteccion_KL.r”**
  - Gráficos de divergencia de Kullback-Leibler calculada para cada instante de tiempo de la serie en estudio.

Antes de utilizarlos se recomienda modificar la información sobre la ubicación de los archivos .csv que contienen los datos recolectados, así como también los tiempos de inicio y término de cada serie en estudio siguiendo como guía los comentarios incluidos dentro de cada uno de los *scripts*.

Para ejecutar el procesamiento de los datos se deben seguir los siguientes pasos:

1. Abrir RStudio.
2. Abrir *script* ”Modelo_prediccion.r”.
3. Modificar campos correspondientes a ubicación de archivo de datos recolectados y tiempos de inicio y término de las series a estudiar.
4. Ejecutar *script* completo o instrucciones aisladas según sea necesario para generar la estadística básica y el modelo ARIMA correspondiente.
5. Utilizando el archivo anterior quedarán los datos importados en el espacio de trabajo de RStudio, por lo tanto para el siguiente paso solo es necesario abrir el archivo ”Deteccion_KL.r” y ejecutar la totalidad de las instrucciones para analizar las series de tiempo con la divergencia de Kullback-Leibler.

C.6. Datos de Acceso a Máquinas Virtuales

1. Cliente tradicional
   - Nombre de usuario: Host1
   - Contraseña: password
2. Cliente Atacante
   - Nombre de usuario: root
   - Contraseña: osboxes.org
3. Enrutadores
   - Nombre de usuario: router
   - Contraseña: router
4. Firewall
   - Nombre de usuario: firewall
   - Contraseña: firewall
5. Máquina de Análisis
   - Nombre de usuario: Sonda
   - Contraseña: password
6. Servidor Web
   - Nombre de usuario: msfadmin
   - Contraseña: msfadmin