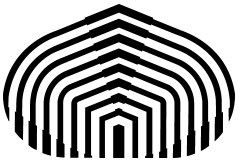


Universidad Simón Bolívar
Decanato de Estudios Profesionales
Coordinación de Ingeniería de la Computación

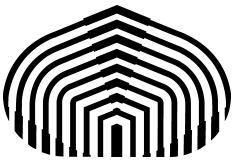
MÓDULO PARA LA DETECCIÓN DE ATAQUE MEDIANTE BRO

Por:
Cordero García Alejandra
Tutor académico:
Álvarez Kity

Tutor industrial:
Díaz Verdejo Jesús

PASANTÍA LARGA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Computación

Sartenejas, Octubre de 2017



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PASANTÍA LARGA

MÓDULO PARA LA DETECCIÓN DE ATAQUE MEDIANTE BRO

Presentado por:
Cordero García Alejandra

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

Kity Álvarez

@jurado1

@jurado2

Sartenejas, @día de @mes de 2017

Resumen

Esta pasantía larga consistió en desarrollar e implementar los módulos necesarios para implementar un detector de intrusiones para servicios web, basado en SSM (por el ingles,"Segmented Stochastic Modelling")[11], que se pueda utilizar en redes de explotación. El detector fué implementado haciendo uso del lenguaje de "scripting" de un sistema de monitorización llamado Bro. Este sistema, se encargará de tomar peticiones HTTP de tipo GET, extraer sus URIs y analizar si los mismas poseen anomalías de una manera probabilística, utilizando modelos de normalidad. En el caso en el que se perciba alguna anormalidad, el sistema se encargará de emitir una alarma. El proyecto estuvo dividido en varias fases: La primera fase consistió en estudiar SSM y sus conceptos relacionados, así como familiarizarse con la herramienta Bro; la segunda fase se basó en diseñar y dividir las tareas del sistema en tres módulos principales: un modulo de segmentación, un módulo de evaluación y un modulo de entrenamiento; en la tercera fase se realizó la implementación y las pruebas funcionales del detector de intrusiones; en la cuarta fase se realizaron las pruebas operativas al sistema completo haciendo uso de bases de datos con trazas normales y bases de datos con trazas anómalas; y finalmente,en la sexta fase, se estructuró y se redactó la memoria del proyecto.

Palabras clave: SSM, HTTP, GET, URI, Bro.

Agradecimientos

Agradezco en primer lugar a mis padres por darme su apoyo incondicional siempre. También debo darle las gracias a mis amigos por brindarme su ayuda y al Dr. Díaz Verdejo por su paciencia y dedicación.

Índice general

Índice de figuras

Laos **A**ve **S**erpiente

INTRODUCCIÓN

Hoy en dia es muy difícil imaginar el mundo sin el Internet. Esta gran red de redes se ha convertido en una parte importante en la comunicación en el mundo. Un gran porcentaje del uso que se le da al Internet proviene de las aplicaciones web. Para compartir toda la información brindada por estas aplicaciones a través del Internet, se hace uso del protocolo HTTP ("Hypertext Transfer Protocol"), un protocolo cliente-servidor. Los servidores HTTP, son los encargados de suplir las peticiones de sus clientes y enviar o recibir la información por los mismos. Este tipo de servidores son un punto apetecible para los hackers ya que los mismos contienen informacion importante de las aplicaciones web. Como acceso a la base de datos de la aplicación que contiene las claves, los nombres de usuarios y número de tarjetas de crédito. Por esta razón, es de total importancia velar por su seguridad.

Este punto será el tema central en el presente trabajo. En el mismo, explicará la implementación y las bases téoricas de un protocolo desarrollado en la Universidad de Granada llamado SSM ("Segmented Stochastic Modelling") en un lenguaje se "scripting"del analizador de redes Bro.

El objetivo de implementar este sistema es el de crear un detector de instrusiones (IDS) que detecte de manera adecuada intrusiones en servidores de tipo HTTP.

La estructura del siguiente trabajo será la siguiente: En el capítulo uno se hablará sobre el planteamiento del problema, los objetivos tanto generales como especificos y la planificación que se utilizó para realizar el proyecto. Por otra parte,en el segundo capítulo se tocarán temas relacionados con el estado del arte del proyecto. En primero lugar se explicará la definición de IDS y los tipos de IDS que existen, se explicará la composición de los URI según el RFC (mencionar el

numero del RFC) y se dará una descripción del funcionamiento SSM los modelos que lo conforman. Además, se describirá la herramiento utilizada y el lenguaje de "scripting"que se utilizó para implementar el sistema. En el tercer capítulo se hablará sobre el diseño del sistema. Aquí se describirán los módulos en los que esta dividido el IDS y como es el funcionamiento de cada uno. Los detalles de implementacion de estos se explicarán en el capítulo cuarto. El quinto capítulo tocará el tema de la evaluación y las pruebas. En este capítulo se hablará sobre la base de datos utilizada para realizar las pruebas y los resultados arrojados por las mismas. Además se describirán un poco las pruebas tanto operativas como funcionales realizadas sobre el sistema.

Capítulo 1

EL PROBLEMA

1.1. Planteamiento del problema

Internet con el pasar de los años se ha estado integrando cada vez más en varios aspectos de nuestras vidas. A pesar de todos los beneficios que trae estar tan conectados a esta red de redes, es preciso recalcar igualmente los problemas que acarrea el hecho de estar conectado a esta sin la existencia de medidas de seguridad necesarias.

Los dispositivos, las aplicaciones y los sistemas que utilizamos en nuestro día a día almacenan información valiosa a la que pueden acceder personas no deseadas si no existen las medidas necesarias en las redes para evitar los ciberataques. Cada día más y más información digital existe en el mundo, y mientras crece el volumen de dicha información , aumenta también la cantidad de ciberataques que se realizan [2]. Es por esta razón, que el concepto de ciberseguridad es de vital importancia cuando hablamos del cibermundo. La ciberseguridad, se encarga de proteger redes de comunicación, sistemas y datos de ataques cibernéticos.

Uno de los métodos utilizados en la ciberseguridad son los detectores de intrusiones (IDS, por sus siglas en inglés Intrusion-detection system). Un IDS es detector que procesa información proveniente del sistema o redes de computadores que se desea proteger. Una vez que que detectada una intrusión este lanza

alertas para activar el proceso de auditoría por parte de los administradores de sistema [10].

Por otra parte, uno de los servicios más utilizados en el mundo del Internet y que más contienen información son los servicios web. Estas características hacen que este tipo de servicios sea uno de los más apetecibles para los ciberataques. Se hace evidente entonces, que se requieren medidas en el área de la ciberseguridad dirigidas a estos casos.

Es por esta razón que grupos de investigación se han dedicado a tratar de solventar dicho problema. Tal es el caso del grupo de investigación de la Universidad de Granada que ha desarrollado un IDS para detectar ataques a servidores web llamado SSM ("Segmented Stochastic Modelling") y así evitar problemas como: el reconocimiento, el escaneo de paquetes de red, la obtención y el mantenimiento de acceso al servidor y la modificacion de "logs" que evitan que se rastree el ataque en este tipo de servicios.

Es por ello que resultaría interesante implementar un sistema de intrusiones que ha presentados resultados satisfactorios como es el caso de SSM haciendo uso de una herramienta tan potente y en auge como lo es Bro.

1.2. Objetivos

1.2.1. Objetivos generales

El objetivo del presente trabajo es incorporar la técnica de detección de intrusiones SSM propuesta en [1] en el sistema Bro. Para ello deben usarse las capacidades de scripting de Bro e incorporar las funcionalidades necesarias como un módulo que pueda activarse a demanda para monitorizar las peticiones a los servicios web.

1.2.2. Objetivos específicos

La consecución del objetivo general planteado puede abordarse a partir de los siguientes objetivos específicos:

1. Interceptar las URI para su análisis haciendo uso de Bro para capturar y filtrar la información de los paquetes del protocolo HTTP.
2. Desarrollar un módulo que permita evaluar el índice de anomalía de los URI haciendo uso de las herramientas otorgadas por el lenguaje de scripting de Bro y siguiendo las especificaciones y expresiones pautadas por la técnica de detección de intrusiones SSM.
3. Desarrollar un módulo para la estimación de los modelos de normalidad haciendo uso de las herramientas otorgadas por el lenguaje de scripting de Bro que permita obtener un modelo de normalidad a partir de trafico libre de ataques.
4. Evaluar experimentalmente el funcionamiento del sistema realizando pruebas, tanto funcionales como operativas para verificar el correcto funcionamiento del mismo.

1.3. Justificación e importancia

La intención del proyecto es implementar un detector de intrusiones para servicios web, basado en SSM haciendo uso de la herramiento Bro.

Un Detector de Intrusiones (IDS) es un sistema que monitoriza todas las actividades de un sistema en específico para detectar intrusiones[4]. Si el mismo detecta una intrusión, se generará una alerta para que los administradores del sistema se encarguen de la misma.

Sin embargo, una pregunta válida que se puede realizar es : ¿Para qué molestarse en instalar un IDS si se tiene un “firewall”, filtro de “spam” y contraseñas

de autenticidad? La respuesta es simple: porque las intrusiones siguen ocurriendo. Así como las personas algunas veces olvidan cerrar las ventanas, por ejemplo, también olvidan actualizar el conjunto de reglas del "firewall". [5]

Es por esto, que los IDS conforman un aspecto importante dentro de la seguridad, ya que si alguno de los elementos fallan siempre estará el detector de intrusiones para generar alertas sobre posibles ataques.

Por otra parte, SSM es un sistema basado en el uso de modelos realizados a partir de las peticiones hechas al servicio web. Los resultados experimentales en este sistema muestran la eficacia del mismo detectando anomalías en este tipo de servicios. Sin embargo, la implementación disponible de SSM es de laboratorio y funciona sobre trazas estáticas, por lo que la técnica no está disponible actualmente para su uso en redes en explotación.

No obstante, para que realizar una implementación de sistema que funcione en redes en explotación, se requieren de herramientas aptas que permitan capturar y analizar el tráfico en la red. Por su parte, Bro, es un sistema de monitorización de red de código abierto muy utilizado hoy en día para el desarrollo de mecanismos de seguridad que posee: su propio lenguaje de scripting y una comunidad de asistencia que va creciendo cada vez más con el tiempo.

Por lo tanto, implementar una técnica que ha tenido resultados satisfactorios como lo es SSM, haciendo uso de Bro, que es una herramienta en auge en el ámbito de la seguridad de redes hoy en día resulta de mucha utilidad ya que esta unión permitirá de manera satisfactoria detectar intrusiones en los servicios web en producción para de esta manera alertar en la manera de los posible sobre ataques de reconocimiento, escaneo, la obtención y el mantenimiento de acceso al servidor y la modificación de "logs" que evitan que se rastree el ataque.

1.4. Planificación

La tareas en las que se divide el desarrollo de este proyecto fueron las siguientes:

1. Estado del arte

Durante esta etapa se realizará un estudio sobre los IDS, los tipos de IDS y el sistema SSM. Así mismo, se hará una lectura del RFC 3986 (URI) y se investigará sobre el protocolo HTTP.

2. Análisis de Bro

   Durante esta etapa del proyecto se instalará, se estudiará la documentación y se aprenderá a hacer uso de la herramienta Bro y su lenguaje de scripting.

3. Arquitectura modular del sistema

   En esta etapa se realizará el diseño de los módulos principales que conforman el sistema. Esta tarea se basará en dividir el trabajo de cada uno de los módulos en varias funcionalidades y en diseñar las estructuras de datos y el formato de los archivos de texto del sistema.

   Esta tarea estará subdividida en las siguientes subtareas:

   3.1. Diseño del módulo de análisis sintáctico/segmentación de URIS

   3.2. Diseño del módulo de evaluación de URIs

   3.3. Diseño del módulo de entrenamiento

4. Implementación del sistema

   En esta etapa del proyecto se llevará a cabo la implementación del sistema haciendo uso del lenguaje de scripting de Bro.

   4.1. Módulo de análisis sintáctico/segmentación de URIS

   En esta etapa se tomarán los diseños esbozados previamente y se implementarán todas las funcionalidades necesarias para realizar el segmentador y el analizador sintáctico de los URIS.

   4.2. Módulo de evaluación de URIs

   En esta etapa se implementarán todas las expresiones, las estructuras de datos y las funcionalidades necesarias para construir un modulo de evaluación funcional haciendo uso de las herramientas presentadas por Bro.

   4.3. Módulo de entrenamiento

   En esta etapa se implementarán las estructuras de datos y módulo de entrenamiento.

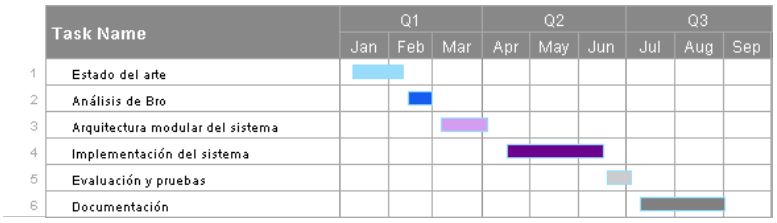


FIGURA 1.1: Diagrama de Gantt del proyecto.

4.4. Módulo de gestión del tipo de sistema

En esta etapa del proyecto se implementará el módulo de gestión del tipo de sistema haciendo uso de "bash scripting".

4.5. Módulo de reportes

En esta etapa se implementarán los reportes del sistema haciendo uso de herramientas de Bro dedicadas exclusivamente a realizar este tipo de tareas.

5. Evaluación y pruebas

Durante el desarrollo del proyecto se realizarán pruebas funcionales y una vez finalizada la implementación del sistema se procederá a realizar pruebas operativas.

5.1. Realización de pruebas funcionales

Se harán pruebas de manera individual a cada una de las funciones más importantes de los módulos que conforman el sistema y se realizarán las modificaciones pertinentes en las mismas.

5.2. Realización de pruebas operativas

La realización de las pruebas operativas se basará en tomar base de datos de trazas tanto normales como anormales, probar el funcionamiento de todo el sistema y realizar las modificaciones pertinentes.

6. Documentación Esta fase consistirá en estructurar y redactar la memoria del proyecto.

Capítulo 2

ENTORNO EMPRESARIAL

A continuación se dará una breve explicación de La Escuela Técnica Superior de Ingenierías Informática de la Universidad de Granada. Lugar en donde se desarrolló el proyecto que se explica en el presente trabajo.

2.1. Descripción de la Empresa

La Universidad de Granada fue fundada en 1531, siendo continuadora de una larga tradición docente que enlaza con la de la Madraza del último Reino Nazarí.

La Universidad está muy presente en la ciudad de Granada, disfrutando de la peculiar belleza de su entorno y de una situación geográfica privilegiada.

En Granada hay cuatro Campus Universitarios, además del "Campus Centro", en el que se integran todos los centros dispersos por casco histórico de la ciudad. Hay otros dos Campus de la UGR en las ciudades de Ceuta y Melilla, en el norte de África.

En la UGR estudian más de 60.000 alumnos de grado y posgrado y otros 10.000 realizan cursos complementarios, de idiomas, de verano, etc. Imparten docencia 3.500 profesores y trabajan más de 2.000 administrativos, técnicos y personal de servicios.[6]

Dentro de la UGR se encuentra la ETSII o Escuela Técnica Superior de Ingenierías Informática, que es un centro docente de la Universidad de Granada situado en el campus Aynadamar, junto a la Facultad de Bellas Artes.[7]

2.2. Misión

La UGR, creada en 1531, es una universidad pública, abierta, conectada con su entorno y con vocación internacional, comprometida con la innovación, el progreso y el bienestar social mediante la mejora continua de la docencia y una investigación de calidad, la extensión y difusión de la cultura y la transferencia del conocimiento. La UGR se orienta por los valores de respeto a la dignidad y libertad de las personas, a la justicia, a la igualdad, a la solidaridad y a la corresponsabilidad en el desarrollo sostenible.[8]

2.3. Visión

La UGR aspira a: [8]

- Ser una Universidad bien valorada por las personas y grupos a los que se orienta tanto externos (alumnos potenciales, agentes sociales, organizaciones públicas y privadas) como internos (estudiantes, personal docente e investigador y personal de administración y servicios).
- Tener un proyecto ético e inteligente que contribuya a un entorno y un mundo mejores, respondiendo y aportando soluciones a las necesidades sociales, culturales, económicas y medioambientales.
- Distinguirse como una Universidad que aprende, con una formación e investigación de calidad reconocida, dinámica e innovadora.
- Ser una institución abierta al saber, la innovación, la crítica, el debate y la sociedad.

2.4. Estructura Organizacional

La Escuela Técnica Superior de Ingenierías Informática está subdividida en cuatro departamentos: Arquitectura y tecnología de computadores, Ciencias de la computación e inteligencia artificial, Lenguajes y sistemas informáticos y Teoría de la señal, telemática y comunicaciones.[9] El departamento en donde se realizó el proyecto fué el departamento de Teoría de la señal, telemática y comunicaciones.

Capítulo 3

MARCO TEÓRICO

En el capítulo se describirán algunos conceptos importantes para comprender la implementación del sistema en el que se basa el presente trabajo. Dichos conceptos son: *IDS, el protocolo HTTP, URI , automátas de estados finitos, automátas probabilísticos, modelo de Markov,SSM y Bro.*

En líneas generales, se tiene como propósito explicar el funcionamiento de un *IDS* basado en anomalías que haga uso del sistema *SSM* y del lenguaje de scripting de la herramienta *BRO*.

Este sistema analiza los *URI* de las peticiones tipo GET o HEAD del protocolo *HTTP*, haciendo uso de la técnica *SSM*. Esta técnica, a su vez hace uso de *autómatas de estados finitos probabilísticos* y del *modelo de Markov* para determinar si las peticiones realizadas a un servidor *HTTP* son posibles ataques o no.

3.1. Protocolo HTTP

El protocolo HTTP, ("Hypertext Transfer Protocol") es un protocolo de la capa de aplicación que se encuentra en el corazón de la Web. Se define en [RFC 1945] y [RFC 2616]. HTTP esta implementado en dos programas: un programa cliente y un programa servidor. El programa cliente y el programa servidor se ejecutan en diferentes sistemas y se comunican entre si intercambiando mensajes HTTP.

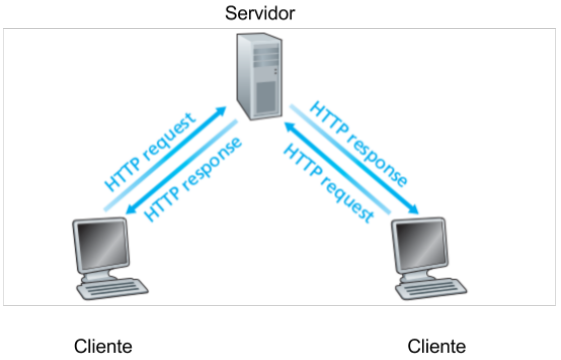


FIGURA 3.1: Petición-respuesta.[24]

HTTP por su parte, define la estructura de dichos mensajes y como se realiza el intercambio de los mismos. [24]

El protocolo HTTP, es un protocolo simple de petición-respuesta que funciona normalmente sobre la capa de transporte TCP. La idea general de la interacción entre el cliente y el servidor se muestra en la figura 3.1.

En el protocolo HTTP, el cliente inicia la conexión TCP con el servidor. Una vez establecida la conexión, ambos podrán enviar y recibir mensajes de petición y respuesta. La información que se transfiere mediante este protocolo pueden ser archivos de texto plano, hipertexto, audio, imágenes o cualquier información accesible por Internet.

Es importante tener en cuenta que el servidor envía los archivos solicitados a los clientes sin almacenar información del estado de los mismo, es decir, el servidor no almacena información de sus clientes. Debido a esto se dice que HTTP es un protocolo sin estado.[24]

Por otra parte, los mensajes de petición y respuesta del protocolo HTTP poseen una estructura general como la que se muestra en la figura 3.2.

La linea de petición en los mensajes HTTP, presenta tres partes separadas por espacios:

1. El método.

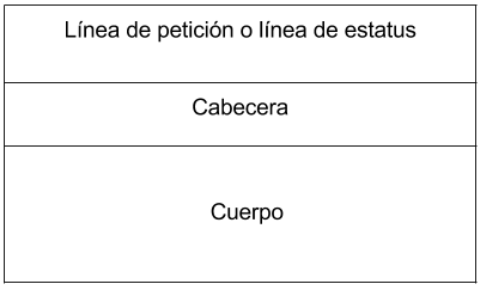


FIGURA 3.2: Mensaje HTTP.[24]

2. El identificador del recurso solicitado (URI).

3. La version del protocolo HTTP.

El método es un parámetro que indica lo que actualmente se esta solicitando por el mensaje. Los métodos GET, HEAD y POST son los más utilizados en las comunicaciones del protocolo HTTP.

- GET se encarga de hacer solicitudes de recursos al programa servidor HTTP.
- HEAD solicita recursos de la misma forma en la que lo hace el método GET con la única diferencia de que la solicitud solo será respondida con la cabecera del recurso, excluyendo el cuerpo del mismo.
- POST solicita que el recurso de destino procese la información que viene incluida en la solicitud realizada

Para el presente trabajo la sintaxis de los URI y los mensajes de petición de tipo GET resultan de importante relevancia.

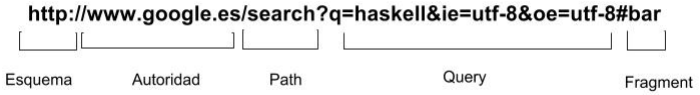


FIGURA 3.3: URI.

3.2. URI

Un URI es una serie de caracteres que identifican un recurso en la red. Este posee una sintaxis específica que está conformada por diferentes segmentos de caracteres: el esquema, la autoridad, la ruta, el "query" y el "fragment".

En la figura 5.7, se muestra, en un ejemplo de los componentes que forman un URI.

A continuación se describe de manera breve cada uno de los componentes que pueden conformar un URI.

- Esquema:

  El esquema identifica el protocolo. En el caso del presente trabajo, los URIs utilizaran únicamente el protocolo http o https.

- Autoridad:

  Este componente del URI posee información del "host" del URI, el cual viene representado bien sea por un IPv4 o un nombre de dominio, seguido de un número de puerto (opcional).

- "Path": La ruta de un URI es un conjunto de segmentos organizados de manera jerárquica que contienen información sobre la ubicación de los recursos a solicitar.

- "Query":

  El "query" de un URI es un segmento de información no jerarquizada, cuyo símbolo de inicio es el signo de interrogación (?). Por lo general, el "query" está conformado por una dupla "atributo=valor" que junto con la ruta identifican el recurso que se desea solicitar. A diferencia de la ruta, la información

aquí contenida debe ser procesada por el servidor al que se le está solicitando el recurso.

- "Fragment":

  En un URI, el "fragment" corresponde a la dirección de un segundo recurso dentro del primer recurso identificado por la ruta y el "query". Esta cadena de caracteres esta precedida por el símbolo de un numeral (#).

En un URI, tanto el esquema como la ruta son segmentos que deben existir de manera obligatoria. Si la ruta es un carácter vacío, se asumirá que la misma es "/". El resto de los segmentos son opcionales.

3.3. Detector de intrusiones (IDS)

Un detector de intrusiones (IDS del inglés Ïntrusion-detection system") es un sistema que se encarga de procesar la información entrante de un sistema a proteger con la intención de detectar actividades maliciosas y lanzar alertas para activar el proceso de auditoría por parte de los administradores de sistema [10]

Los IDS suelen clasificados en: IDS basados en firmas o IDS basados en anomalías.

Los IDS basados en firmas, utilizan una base de datos de ataques conocidos y vulnerabilidades del sistema los cuales son utilizados para realizar comparaciones con los patrones seguidos por las actividades detectadas. Si se detecta similitud con alguno de los patrones de la base de datos, se activará una alarma para indicar que un ataque esta siendo perpetuado.

La eficacia de los detectores de intrusiones basados en firmas es muy buena, sin embargo, su buen funcionamiento depende completamente de la constante actualización de la base de datos de ataques.

Por otra parte, los IDS basados en anomalía detectan los ataques de manera diferente. En este tipo de IDS los ataques son detectados a partir de la observación de comportamientos, bien sea del sistema, o de los usuarios.

El modo de funcionamiento de un IDS basado en anomalía se basa en: recolectar la información de las actividades detectadas y comparar dicha información con un modelo de normalidad del sistema que ha sido previamente construido. El modelo de normalidad se construye a partir de comportamientos previamente observado en el sistema y que son catalogados como "normales". Si existe una incongruencia entre la información entrante y el modelo de normalidad, entonces se generará una alarma.

No obstante, uno de los problemas de este tipo de detector de intrusiones es la alta tasa de falsos positivos que puede llegar a tener. Esto se debe, a que los modelos, por lo general, no contienen todos los comportamientos normales de un sistema en su totalidad. Esto, a su vez provoca que existan eventos libres de ataques que sean catalogados como una amenaza. También está el hecho de que es muy posible que con el pasar del tiempo, el comportamiento del sistema a proteger vaya cambiando, lo cual implicaría que, si no se hace una constante revisón y reentrenamiento del modelo de normalidad el mismo quedará obsoleto y los eventos entrante serán mal catalogados por el IDS. El problema de realizar entrenamientos constantes para actualizar un modelo de normalidad es que se crearán brechas de tiempo en donde el sistema será muy vulnerable a ataques ya que en lugar de estar funcionando el módulo para detección, estará funcionando el módulo para entrenar el modelo de normalidad. Por lo tanto, si un ataque es perpetrado durante ese periodo de tiempo, el mismo quedará grabado en el modelo de normalidad como un comportamiento normal del sistema.

También existen los IDS híbridos cuyo funcionamiento mezcla el funcionamiento de los detectores de intrusiones basados en firmas y los basados en anomalía, es decir, este tipo de IDS suelen contar con una base de datos de firmas y también con un modelo de normalidad.

3.4. Autómatas de estados finitos

Un autómata es un modelo abstracto de un ordenador digital [13]. Una manera de definirlo de forma gráfica es como se muestra en la figura 3.4.

donde [14]:

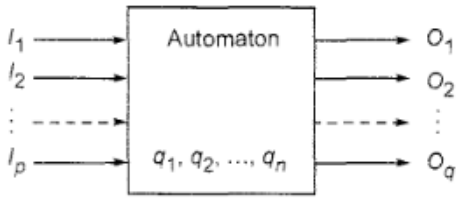


FIGURA 3.4: Autómata.[14]

- $I_1, I_2, ..., I_n$ es el ïnput"del autómata que es introducido en cada instante de tiempo $t_1, t_2, ..., t_n$.
- $O_1, O_2, ..., O_n$ es la salida del autómata.
- $q_1, q_2, ..., q_n$ son los estado en los que puede estar el autómata en cualquier instante de tiempo.

Dentro de los autómatas existe la familia de los autómatas de estado finito. Un autómata de estado de autómata finito se puede definir formalmente como una quíntupla [15]:.

$$M = (Q, \Sigma, \delta, q, F) \tag{3.1}$$

donde:

- Q es un conjunto finito de estados.
- Σ es un conjunto finito, llamado alfabeto. Los elementos de Σ se llaman símbolos.
- δ es una función de transición, tal que $Qx\Sigma \leftarrow Q$
- q es un elemento perteneciente a Q llamado estado inicial.
- F es un subconjunto de Q y representa el conjunto de estados finales.

Los autómatas finitos se clasifican en autómatas finitos deterministas (DFA, del inglés "Deterministic Finite Automata") y no-deterministas (NFA, del inglés "Nondeterministic Finite Automata").

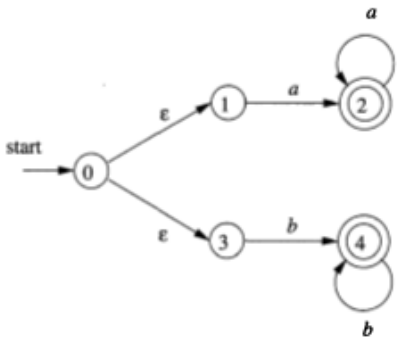


FIGURA 3.5: Autómata no-determinista.[15]

Los autómatas finitos deterministas se definen teóricamente de la misma manera en la que se definen los autómatas finitos. Sin embargo, se debe cumplir que por cada estado "s" y símbolo de entrada "a" debe existir un único estado al que se puede hacer una transición. Esto quiere decir que la función de transición δ de los autómatas finitos deben retornar un solo elemento.

Por otra parte, los autómatas finitos no-deterministas al igual que los autómatas finitos deterministas se definen teóricamente de la misma forma que los autómatas finitos. No obstante, la función de transición δ de estos autómatas puede retornar un conjunto de elementos, es decir, por cada estado "s" y símbolo de entrada "a" pueden existir varios estados al los que se puede hacer una transición.

Tanto los autómatas finitos deterministas como los no deterministas se puede representar a tráves de un grafo. En donde los estados serian representados a tráves de los nodos y las aristas representarían las funciones de transición.

Un ejemplo de un NFA representado a tráves de un grafo se presenta en la figura 3.5.

En la figura 3.6 se muestra la representación de un DFA haciendo uso de un grafo.

Los autómatas deterministas permitirían determinar, dado un autómata asociado a un protocolo, si una secuencia de mensajes intercambiados puede ser decodificada mediante dicho autómata. Esto es, si la secuencia de mensajes corresponde a la especificación del protocolo y, por lo tanto, es gramaticalmente correcta. No

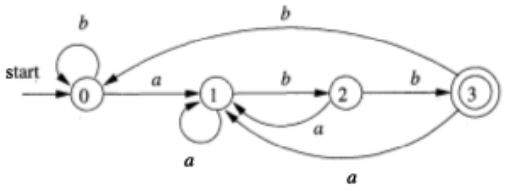


FIGURA 3.6: Autómata determinista.[15]

obstante, una parte importante de los ataques son estructuralmente correctos de acuerdo al protocolo, es decir, siguen la gramática [18].

Sin embargo, los autómatas no deterministas permitirán discriminar ataques cuya estructura sintáctica sea correcta porque permiten incorporar información relacionada con la semántica o el contexto .

Un tipo de autómata no determinista de especial interés para los objetivos del presente trabajo son los autómatas de estado finitos probabilístico [Brookshear, 1989]. En estos se consideran probabilidades asociadas a los estados y/o a los símbolos de forma que se atribuye una naturaleza probabilística tanto a la secuencia de estados que sigue el sistema como a la de los símbolos observados.

3.5. Modelo de Markov

En un proceso de Markov un evento en un tiempo determinado, depende de los procesos inmediatos anteriores a este [16]. Un modelo de Markov discreto, por otra parte, consiste en un conjunto de estados y una serie de probabilidad de transición que rigen las transiciones entre estados y la producción de símbolos.

Un modelo de Markov discreto, λ, se define como un quíntupla:

$$\lambda = (Q, \Theta, A, B, \Pi) \tag{3.2}$$

donde:

- Q : Es el conjunto de N estados del modelo.
- Θ : Es el vocabulario del modelo, es decir, son los posibles símbolos o eventos observables del sistema.
- A : Es una matriz NxN de probabilidades de transición entre estados.

$$\begin{aligned} A = [a_{ij}], 1 \leq i \leq N y 1 \leq j \leq N. \\ a_{ij} = P(q_t = S_j | q_{t-1} = S_i) \end{aligned} \tag{3.3}$$

- B : Es una matriz NxM de probabilidades de generación u observación de los símbolos .

$$\begin{aligned} B = [b_{ik}], 1 \leq i \leq N, 1 \leq k \leq M. \\ b_{ik} = P(O_t = v_k | q_t = S_i) \end{aligned} \tag{3.4}$$

- Π es el vector de probabilidades del estado inicial.

$$\begin{aligned} \Pi = [\Pi_i], 1 \leq i \leq N \\ \pi_i = P(q_1 = S_i) \end{aligned} \tag{3.5}$$

Un ejemplo sencillo de un modelo de Markov sería el siguiente [17]

S = s_1,s_2, donde, s_1 = "*lluvioso*" y s_2 = "*soleado*" y la matriz de transición es:

$$P = \begin{pmatrix} 0{,}75 & 0{,}25 \\ 0{,}25 & 0{,}75 \end{pmatrix}$$

Un modelo de Markov es un autómata de estados finitos no determinista que puede ser representado mediante grafos.

El ejemplo presentado con anterioridad, por lo tanto se puede graficar de la manera en la que se muestra en la figura 3.7.

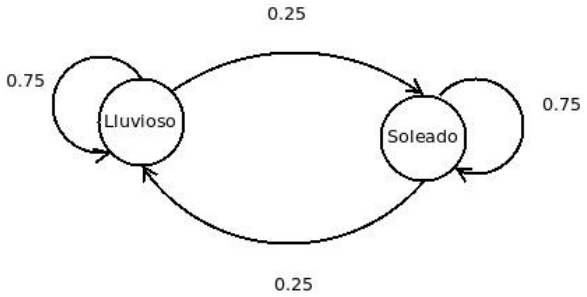


FIGURA 3.7: Ejemplo modelo Markov.

3.6. SSM

SSM o *Structural Stochastic Modeling* en inglés es una tecnica desarrollado por Estevez-Tapiador. A grandes rasgos, esta técnica se basa en la definición de un autómata de estados finitos estocástico capaz de evaluar la probabilidad de generación de una petición concreta. El autómata permitirá, por tanto, dada una petición, evaluar si dicha petición es legítima (corresponde al modelo) y su probabilidad. En función de la probabilidad y de un umbral se clasificaran las peticiones como normales o anormales.[1]

Esta técnica, se basa en la teoría de los modelos de Markov ya que se define un autómata de estados finitos que permite evaluar la probabilidad de generación de una petición, dado un modelo previamente construido.

Por otra parte, en el presente trabajo, las peticiones que serás estudiadas por dicha técnica serán peticiones de tipo GET del protocolo HTTP. En concreto, de las peticiones GET el elemento a estudiar serán los URIs de dichas peticiones. No obstante, se puede extender para que funcione con métodos de tipo POST y HEAD.

Este hecho hace que el autómata de SSM deba tomar una cierta topología para que de esta forma se puedan reconocer los URIs de cada petición. Dicha topología se infiere a tráves de la sintáxis de los URIs. En la figura 3.8 se muestra la topología del autómata.

Las transiciones del autómata vienen dada, por las especificaciones de las sintaxis de los URIs, que están descritas en el RFC 3986. Además, se puede observar

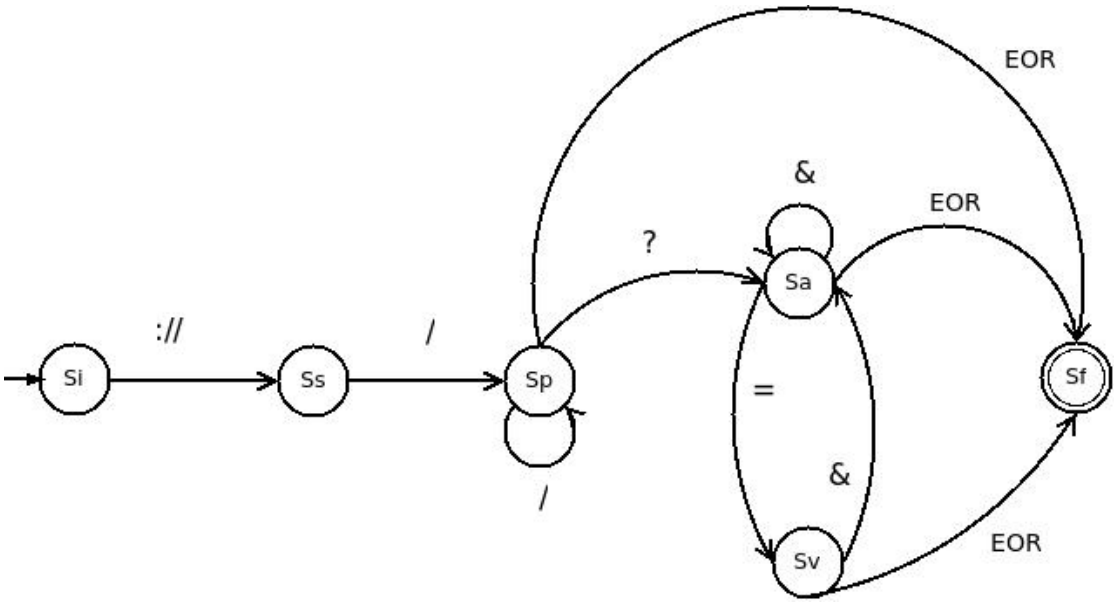


FIGURA 3.8: Automata del modelo SSM

en la figura 3.8 que el autómata tiene un estado inicial (S_I), un estado del "host" (S_S), un estado de segmento de ruta (S_P), un estado atributo (S_A), un estado valor (S_V), un estado final (S_F) y un estado sumidero (S_{OOS}) en el que se termina si el URI no es sintácticamente correcto.

Si una petición del protocolo HTTP anexa un URI que no puede ser reconocido por el autómata descrito con anterioridad implicará que el URI de dicha petición no está bien construido sintácticamente.

Además, existirá un vocabulario diferente para los estados S_S, S_P, S_A y S_V. Estos vocabularios, son construidos, a partir de tomar múltiples peticiones libres de ataques realizadas al servidor.

Entonces, en resumen, lo que hace la técnica de SSM para detectar intrusiones es verificar la sintaxis del URI de las peticiones enviadas al servidor a tráves del autómata descrito con anterioridad y, al mismo tiempo, se estudia segmento por segmento del URI la probabilidad que tiene cada palabra de aparecer en un estado determinado del autómata, para así verificar si el URI que viene anexado al la petición es una cadena de caracteres que probabilísticamente corresponde a un petición normal o no.

Teóricamente, el modelo utilizado por SSM es el siguiente [18]:

- Q: es el conjunto de estados (S_I,S_S,S_P,S_A,S_V,S_F,S_{OOS}).
- θ: es el conjunto de símbolos observables que se encuentran en el vocabulario de cada estado.
- A: es la matriz de probabilidad de transiciones entre estados
- B: es un conjunto de vectores que contiene la probabilidad de las palabras observadas en cada estado. (B_I,B_S,B_P,B_A,B_V,B_F,B_{OOS}).
- Π: es el vector de probabilidades iniciales, cuyos valores están determinados por la topología del modelo.

Evaluación

Dado un conjuntos de observaciones $0 = o1, o2, ..., oT$. Cada una pertenecientes a los estados $Q = q1, q2, .., qT$. En principio, se puede realizar la evaluación de un URI, dado un modelo λ, calculando la probabilidad del conjunto completo de observaciones O del modelo λ siguiendo el patrón de la secuencia de estados Q. Es decir:

$$P(O|\lambda, Q) = \pi_{q1} b_{q1o1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}} b_{q_{t+1} o_{t+1}} \tag{3.6}$$

Para evitar problemas de desbordamiento de calcula el logaritmo de la probabilidad en lugar de la probabilidad.

$$log(P(O|\lambda, Q)) = log(\pi_{q1} b_{q1o1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}} b_{q_{t+1} o_{t+1}}) \tag{3.7}$$

Si se considera que las probabilidades iniciales son cero excepto para S1 y que las probabilidades de transición son equivalentes a 1. Entonces, se tendría que la fórmula presentada en 3.7 se podría resumir en la siguiente:

$$P(O|\lambda, Q) = \sum_{t=1}^{T} b_{q_t o_t} \tag{3.8}$$

Entonces, en primera instancia se podría decir que el índice de anormalidad de un URI U, dado un modelo λ, se puede obtener de la siguiente forma:

$$N_s = -log(P(U|\lambda)) \tag{3.9}$$

La fórmula 3.9 será aplicada sólo si el URI U llega al estado final Sf, de otra manera al índice de anormalidad Ns se le será asignado ∞, es decir:

$$N_s = \begin{cases} -log(P(U|\lambda)) & \text{si } q_t = S_f \\ \infty & \text{si } q_t \neq S_f \end{cases} \tag{3.10}$$

Como se puede observar en la fórmula 3.10, mientras menor sea la probabilidad de aparición de las observaciones, mayor será el índice de anormalidad, por esta razón, una vez calculado el índice, se podría decir que un URI U es anómalo, si el índice de anormalidad es mayor o igual a un umbral de detección θ de lo contrario se dirá que el URI no posee anomalías, esto es:

$$Clase(U) = \begin{cases} Normal & \text{si } N_s(U) \leq \theta \\ Anomalo & \text{si } N_s(U) > \theta \end{cases} \tag{3.11}$$

El umbral de detección es un parámetro que se calcula de forma experimental. Durante la pruebas se busca conseguir un valor θ con el cual se pueda obtener la proporción óptima entre las detecciones de anomalías correctas y los falsos positivos.

No obstante, la fórmula presentada en el apartado 3.10 posee algunos inconvenientes ya que no se estipula el caso en el que existan problemas de entrenamiento insuficiente que será presentado en la sección o que los diferentes vectores B, tengan diferente número de observaciones. El segundo caso provoca un inconveniente

a la hora de realizar la evaluación ya que el acumulado de las probabilidades de los vectores B estarán ligados a la longitud de los mismos. Para solucionar dicho problema se utilizara la propuesta de Estévez [Estévez-Tapiador 2004a], en el que se establece como factor de compensación la probabilidad de observación para de esta manera obtener una probabilidad normalizada. Esta sería la siguiente expresión:

$$\varepsilon_0 = E[B] = \frac{1}{M}\sum_{i=1}^{M} b_i \tag{3.12}$$

Por otra parte, el entrenamiento insuficiente es un problema que se presenta cuando se toma una palabra durante la evaluación que no pertenezca al conjunto de palabras observadas durante el entrenamiento en el modelo. Esto puede ser debido a que la palabra es una palabra anómala o que no se entrenó lo suficiente al sistema. En este caso, la solución que se le aplicará a dicho problema será la de asignar un valor fijo de probabilidad muy baja a estos casos (poov(S_i): probabilidad fuera de vocabulario). Cada uno de los estados del modelo poseerá valores independientes de probabilidades que serán utilizados una vez se detecte este problema.

El otro caso en el que se utilizarán los valores de poov(S_i), es cuando la probabilidad de una palabra que está en el vocabulario es inferior al poov(S_i), es decir:

$$p_{qtot} = \begin{cases} b_{qtot} & \text{si } o_t \in O \text{ y } b_{qtot} > p_{oov}(q_t) \\ p_{oov} & \text{en otro caso} \end{cases} \tag{3.13}$$

Entonces, la nueva expresión del índice de anormalidad que toma en cuenta todas las consideraciones nombradas con anterioridad, es la siguiente:

$$N_s = \begin{cases} -Tlog(\varepsilon_0)\sum_{t=1}^{T} log(p_{qtot}) & \text{si } q_t = S_f \\ \infty & \text{si } q_t \neq S_f \end{cases} \tag{3.14}$$

Entrenamiento

Dada una secuencia de observaciones 0 = o1,o2,...,oT ,y su correspondiente secuencia de estados, Q = q1,q2,...,qT.

El entrenamiento requiere de un conjunto considerable de observaciones junto con los estados asociados a cada una de ellas.

Por lo tanto, se considerara un conjunto de entrenamiento ω de L pares de secuencias de estados

$$\omega = (O^1, Q^1), (O^2, Q^2), ..., (O^L, Q^L) \tag{3.15}$$

tal que:

$$\begin{aligned} O^i &= o_1^i, o_1^i, ..., o_{Ti}^i \\ Q^i &= q_1^i, q_2^i, ..., q_{Ti}^i \end{aligned} \tag{3.16}$$

Entonces, para crear el vocabulario del modelo de normalidad solo se tendrá que hacer un recuento de las frecuencias de aparición relativas de los símbolos y los estados, es decir:

$$\theta_{Sk} = \bigcup_{i=1}^{L} o_j^i | q_j^i = S_k \tag{3.17}$$

Por otra parte, para calcular la probabilidades de observación se utilizaría la siguiente expresión:

$$b_{ij} = \frac{\sum_{s=1}^{L} \sum_{t=1}^{Ts} \delta(o_t^s = v_{i,j}, q_t^s = s_i)}{\sum_{s=1}^{L} \sum_{t=1}^{Ts} \delta(q_t^s = s_i)} \tag{3.18}$$

Donde, δ es una función que retorna 1 si todos sus argumentos son verdaderos y 0 cuando son falsos.

Normalización de los URIs

La normalización consiste en tomar el URL en el formato en el que viene y codificarlo al formato de tipo UTF-8. Este paso evitará considerar palabras o frases iguales que estén escritas en formatos diferente como elementos distintos.

Un ejemplo sencillo de lo que se haría en la normalización del sistema sería el siguiente:

Supongamos que al servidor de tipo HTTP recibe dos peticiones de tipo GET con los siguientes URIs:

- https://192.168.0.23?q=security+network
- https://192.168.0.23?q=security %20network

Ambos URIs poseen la misma información, sin embargo, el espacio en blanco esta codificado de manera diferente en ambos casos. Si no se lleva a cabo la normalización, la palabra "security+networkz "security %20network"serían tomadas como dos palabras totalmente diferentes por el sistema. En este caso, la función de normalización se encargaría de traducir el primer URI al formato UTF-8 para que no exista este problema. Por lo tanto,la frase "security+network"pasaría a ser "security %20network".

Segmentación de los URIs

La idea de la segmentación de URI es considerar una serie de delimitadores, que delimitan áreas especificas dentro del URI.

Los delimitadores que se tomarán en cuenta para segmentar serán los asociados al estándar del protocolo HTTP. Estos son los siguientes:

- D1 = ://, delimitador de protocolo.
- D2 = /, delimitador de recursos.

- D3 = ?, delimitador de parámetros.
- D4 = =, delimitador de asignación de atributos.
- D5 = &, delimitador entre parametros.
- D6 = ASCII 32, delimitador de fin de recurso

A modo ilustrativo, se utilizará un ejemplo concreto con un URI para mostrar como funciona la segmentación en el sistema.

Supongamos que tenemos el siguiente URI:

http://159.90.9.166/consulta/?manifestacion=Pinturas+Rupestres

Entonces, la función que se encarga de segmentar tomará dicho URI y lo segmentará en las siguientes partes:

- http : Segmento correspondiente al protocolo del URI.
- 159.90.9.166 : Segmento correspondiente al "host" del URI.
- consulta : Segmento correspondiente al recursos del URI.
- manifestacion : Segmento correspondiente al atributo del query.
- Pinturas %20Rupestres : Segmento correspondiente al valor del query.

Capítulo 4

MARCO TÉCNOLOGICO

En este capítulo se dará una breve explicación de las herramientas utilizadas para implementar y probar el sistema implementado. De igual manera, se detallarán las especificaciones de la estación de trabaja que se utilizó para realizar dicho trabajo.

4.1. Herramientas utilizadas

En esta sección se explicarán las herramientas utilizadas al momento de implementar el sistema de detección de intrusiones.

En primer lugar se detallarán las caracteristicas del sistema de monitorización de red, (*Bro*) cuyo lenguaje de programación fué utilizado para implementar el IDS. Luego, se explicará cual fué el controlador de versiones utilizado en el proyecto (*GitHub*). Finalmente se describirá la herramienta utilizada para realizar las capturas de paquetes de red que fueron utilizadas para efectuar las pruebas sobre el sistema (*Wireshark*).

4.1.1. Bro

Bro es un software "open source" capáz de analizar a detalle las actividades que ocurren en la red. Se utiliza principalmente para inspeccionar todo el tráfico y detectar signos de actividad sospechosa. Sin embargo, Bro soporta una amplia gama de tareas de análisis de tráfico incluso fuera del dominio de seguridad, incluyendo mediciones de rendimiento y "trouble-shouting".[23]

Este software, posee un conjunto extenso de "logs" que registran las actividades que se observan en la red. Estos "logs" almacenan información de la capa de aplicación, como por ejemplo, los URIs y las respuestas del servidor en una sesión de tipo HTTP, o las solicitudes y respuestas DNS en una sesión DNS. Bro escribe todos estos datos en archivos con formatos bien predeterminados con la intención de facilitar el procesamiento de la información almacenada por programas externos. No obstante, el usuario tiene la posibilidad de elegir el formato en el que se almacenan los datos.

Por otra parte, Bro incluye también un lenguaje de "scripting", Turing completo y orientado a eventos, cuya función es extender y personalizar las funcionalidades primarias que otorga el mismo. Este lenguaje de "scripting" posee una biblioteca estándar.

Así mismo, el lenguaje de Bro posee diversos "frameworks" que facilitan el trabajo al momento de programar nuevas funcionalidades. Los "frameworks" utilizados en el sistema implementado fueron: el "Input Framework" y el "Logging Framework". El "Input Framework" permite leer de manera mas cómoda y eficiente la información que se encuentra dentro de los "logs" del sistema. Este "framework" posee dos modalidades. La primera modalidad consiste en leer la información que se encuentra dentro de los "logs" para luego almacenarla en una estructura de datos de tipo "table". La segunda modalidad lee la información y la envía a un evento anteriormente programado.Por otra parte, el "Logging Framework", tiene como tarea facilitar el manejo y la escritura de los "logs". Este, permite crear, escribir datos de forma organizada y filtrar información de los "logs".

Como se mencionó con anterioridad, el lenguaje de "scripting" de Bro es un lenguaje orientado a objetos. Sin embargo, es importante recalcar, que el núcleo

o motor principal de Bro es el motor de eventos, el cual reduce la información entrante de la red, en eventos de alto nivel. Por ejemplo, si se recibe una solicitud de tipo HTTP, Bro se encargará de convertir la misma en un evento de tipo "http_request" en donde se almacenará información como: la dirección IP, los puertos involucrados, el URI y la versión de tipo HTTP en uso.

A su vez, los eventos creados por Bro a partir de la información entrante de la red son enviados a los manejadores de eventos escritos en el lenguaje de "scripting" de Bro. Estos manejadores administran dichos eventos a tráves de una cola ordenada.

En Bro hay eventos primitivos, como es el caso de "http_request" –evento generado por las solicitudes tipo HTTP – , "dns_request", –evento generado por las solicitudes tipo DNS– , "bro_init" – evento generado al momento en el que Bro inicializa– y "bro_done" –evento que se genera cuando se culminan de realizar todas las tareas que se deben realizar–. Sin embargo, este software brinda la capacidad a los programadores de implementar eventos propios. Estos eventos son un tipo especial de función que poseen las características de: no retornar valores, poder ser programados para una ejecución posterior y poseer una prioridad asociada y configurable que determine el orden en el que serán ejecutados.

Así como se pueden escribir eventos en Bro, también se pueden implementar funciones haciendo uso de la palabra reservada "function".

Por otra parte, el lenguaje de Bro además de permitir implementar eventos y funciones, posee diversos tipos y estructura de datos que facilitan el manejo de la información. El conjunto de tipos de datos que posee el mismo se basa en los tipos de datos comunes que encontramos en la mayoría de los lenguajes de programación como: el "int", que representa los números enteros de 64 bits, "double" que representa los números flotantes de 64 bits, "bool" para los booleanos y "count" para los números enteros sin signo de 64 bits, y en un grupo de tipos de datos mas especifico como: el "addr" que representa direcciones ip, "port" para los puertos de la capa de transporte, "subnet" para las mascaras de subred, "time" para almacenar datos que representen tiempo, "interval" para representar intervalos de tiempo y "pattern" para las expresiones regulares.

Las estructura de datos que permite el lenguaje son: los conjuntos ("set"), las tablas ("table"), vectores ("vector") y registros ("record").

4.1.2. GitHub

GitHub es un controlador de versiones basado en Git y un servidor de "hosting". Este ofrece todas las funcionalidades de Git y a su vez, añade sus propias funcionalidades. GitHub provee control de acceso y varias caracteristicas para la colaboración de proyectos como: el seguimiento de errores, manejador de tareas y wikis. [19]

Por otra parte, Git es un programa "open source" para el rastreo de cambios en archivos de texto. Esta herramiento fué escrita por el autor del sistema operativo Linux. [20]

4.1.3. Wireshark

Wireshark el analizador de paquetes de red "open source". Este permite observar lo que esta ocurriendo en la red a un nivel microscopico [21].

Un analizador de paquetes de red captura los paquetes de red e intenta mostrar los datos del mismo de la manera mas detallada posible [22].

Algunos usos que le dan las personas a Wireshark son:

- Los administradores de red lo usan para solucionar problemas en la red.
- Los ingenieros en seguridad lo utilizan para examinar problemas de seguridad.
- Los desarrolladores lo utilizan para encontrar errores en los protocolos implementados
- Las personas lo utilizan para aprender protocolos de red.

4.2. Estación de trabajo

4.2.1. Especificaciones

A continuación, se describirán las especificaciones de la máquina que se utilizó en el proyecto.

- **Procesador**: 7th Generation Intel® Core™ i3 processor.
- **Memoria (RAM)**: 4GB
- **Sistema operativo**: Ubuntu 14.04
- **HDD (principal)**: 500GB

Capítulo 5

DESARROLLO DE PASANTÍAS

5.1. Estado del arte

En esta etapa del proyecto se realizó una investigación sobre los conceptos fundamentales del sistema a implementar. Las definiciones estudiadas fueron: el protocolo HTTP, los URIs, detector de intrusiones (IDS), modelo de Markov y SSM. El resumen de esta investigación se encuentra en el capítulo 3 del presente trabajo.

5.2. Análisis de Bro

En esta etapa del proyecto se descargó, se instaló y se estudio la herramienta Bro y su lenguaje de "scripting". Bro, es la herramienta principal que se utilizó en la implementación del sistema. Un resumen de esta y de su lenguaje de programación se encuentra en la sección 4.1.1.

5.3. Diseño de la arquitectura del sistema

Esta sección tiene la intención de mostrar la visión general del modelado del sistema que se realizó a partir de las bases teóricas. Aquí, se explicará la arquitectura, el diseño y el modo en el que el IDS basado en SSM interactuará con Bro. Ademas, se detallaran las salidas y los datos de configuración que va a considerar el sistema para su correcto funcionamiento.

5.3.1. Arquitectura del sistema

La arquitectura del detector de intrusiones que se muestra en el presente trabajo se basa en una arquitectura modular la cual está conformada por tres modulo. Un modulo para realizar la segmentación de los URIs, otro para realizar la evaluación y un tercero para realizar el entrenamiento.

A grandes rasgos, el módulo de segmentación, se encargará de tomar los URIs previamente filtrados de las solicitudes de tipo HTTP/GET, lo normalizará y lo segmentará de la forma en la que se explicó en el apartado de "segmentación" de la sección 3.6.

Por otra parte, el modulo de evaluación se encargará de evaluar la probabilidad de generación de cada uno de los segmentos del URI generados por el módulo de segmentación para al final decidir si el URI de la solicitud enviada al servidor web es anómalo o no.

Por último, el modulo de entrenamiento será el encargado de crear el modelo de normalidad del sistema. Para esto, el sistema recibirá solicitudes libres de ataques e irá calculando la probabilidad de aparición de cada una de las palabras que aparecen en las mismas.

Tanto el modulo de evaluación, como el modulo de entrenamiento son dependientes del modulo de segmentación ya que requieren de los segmentos de URI generados por eso para realizar su trabajo.

La arquitectura del sistema queda detallada en la figura 5.1.

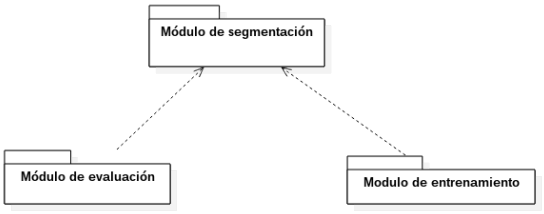


FIGURA 5.1: Arquitectura del sistema.

Así mismo, es importante mencionar que el sistema contará con varios modos de operación que serán explicados a continuación:

- Modo evaluación: En este modo de operación solo se activarán los módulos de segmentación y evaluación del sistema. A manera general, esta modalidad, se encargará de recibir los URI previamente extraídos de las peticiones de tipo HTTP/GET, segmentarlos a tráves del módulo de segmentación, para luego evaluar si el mismo es anómalo o no haciendo uso del módulo de evaluación. El funcionamiento de esta modalidad queda detallado en la figura 5.2.

- Modo entrenamiento "Online": En este modo de operación solo trabajaran los módulos de segmentación y entrenamiento. En esta modalidad el módulo de entrenamiento tomará los segmentos arrojados por el módulo de segmentación, calculará la probabilidad de aparición de los mismos para de esta manera ir modificando un modelo de normalidad previamente establecido. El funcionamiento de esta modalidad queda detallado en la figura 5.2.

- Modo entrenamiento "Offline": Esta modalidad del sistema en análoga al modo de entrenamiento "Online". El único aspecto que diferencia a ambas modalidades es que cuando el sistema funciona en modo "Offline" no se toma en cuenta, ni se modifica un modelo de normalidad previamente construido. La salida de este modo de entrenamiento sera un modelo de normalidad construido desde cero.El funcionamiento de esta modalidad queda detallado en la figura 5.2.

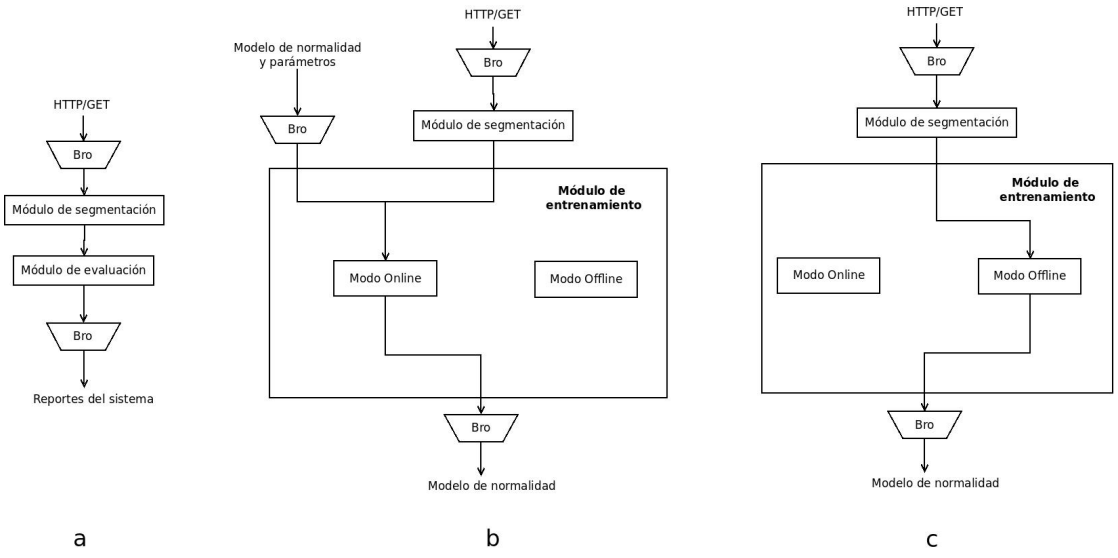


FIGURA 5.2: Modo evaluación (a), modo entrenamiento "Online" (b), modo entrenamiento "Offline" (c).

5.3.2. Filtrado de paquetes HTTP/GET

Esta funcionalidad se encargará de observar los paquetes de la red y filtrar aquellos de tipo HTTP/GET. Una vez obtenido este tipo de paquetes, esta función extraerá el URI adjunto al mismo.

5.3.3. Módulos

En esta sección se describirán las tareas de cada uno de los módulos que conformar la arquitectura detallada en la figura 5.1, el modo en el que se subdividieron dichas tareas, los datos de entrada y salida, y la interacción que existe entre los mismos.

5.3.3.1. Módulo de segmentación

El modulo de segmentación es un elemento clave dentro de la construcción del IDS basado en SSM, es por eso que su buen diseño e implementación es importante para el buen funcionamiento del sistema. Este se encarga, como se mencionó anteriormente,de tomar el URI que proviene de la solicitud de tipo HTTP/GET que se

le hace al servidor HTTP capturado por Bro, normalizarlo y segmentarlo siguiendo las especificaciones que se explicaron en la sección 3.6.

Es evidente entonces, que este modulo consta de dos funcionalidades fundamentales: la normalización de los URIs y la segmentación. La normalización se encargará de tomar el URI de las peticiones HTTP/GET entrantes y codificarlo a formato UTF-8. Normalizar es un paso importante dentro del sistema ya que se estandarizar la forma en las que están escritos los URIs facilita tanto la evaluación como el entrenamiento en el sistema. Por otra parte, la salida arrojada por esta función de normalización será tomada por la de segmentación, quien a su vez se encargará de segmentar el URI de la forma en la que se explica en la sección 3.6, es decir, el URI se dividirá en las diferentes partes estipuladas en el RFC 3986: el "host", la ruta, los argumentos, los valores y el "fragment".

En las base teórica, la segmentación de los URIs se realiza mediante un autómata que se encarga de reconocer (realizar un análisis sintáctico) y evaluar en cada uno de sus estados la probabilidad de generación de cada uno de los segmentos. No obstante, en la función de segmentación del sistema implementado, esta tarea se modeló mediante un analizador sintáctico que hace uso de una gramática (libre de contexto) de atributos que genera el mismo lenguaje que reconoce el autómata presentado en la figura 3.4, es decir, el lenguaje de los URI.

En la figura 5.3 se puede apreciar el diagrama de bloques que refleja el funcionamiento del módulo de segmentación.

5.3.3.2. Módulo de evaluación

El módulo de evaluación, es el encargado de evaluar la probabilidad de generación de cada uno de los segmentos del URI otorgados por el modulo de segmentación, dado un modelo de normalidad. Una vez calculadas estas probabilidades, el modulo se encargará de calcular un índice de anormalidad del URI mediante el uso de las formulas descritas en la sección 3.6 para luego compararlo con un parámetro θ (3.11) y de este modo saber si los segmentos de URI que ingresaron como entrada posee alguna anormalidad o no. Luego de realizar esta evaluación, el modulo se encargara de escribir los reportes del sistema en un "log".

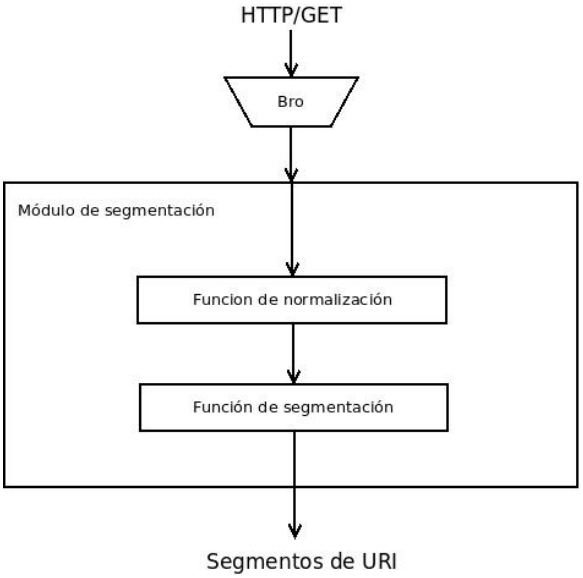


FIGURA 5.3: Diagrama de bloques del módulo de segmentación.

El módulo de evaluación, está conformado por dos grandes funcionalidades: una que se encargará de leer el modelo de normalidad y otra que calculará el índice de anormalidad y evaluará si el mismo es anómalo o no.

El modelo teórico que fue explicado en la sección 3.6 posee elementos como: un conjunto "S" de estados, un conjunto "O" de símbolos observables que se encuentran en cada estado, una matriz "A" que contiene la probabilidad de transición entre estados, un conjunto "B" de vectores que contiene la probabilidad de las palabras observadas en cada estado y un vector de probabilidades iniciales. No obstante, los únicos elementos del modelo que se necesitan ingresar en el sistema para que este funcione y realice las tareas de evaluación son: el conjunto "O" de símbolos observables que se encuentra en cada estado y el conjunto de vectores "B". El conjunto "B" es necesario para resolver la expresión 3.12. Por otra parte, es necesario hacer uso tanto de "B" como "O" para obtener el valor de b_{qtot} presentado en la ecuación 3.13. Este elemento es sumamente importante para calcular el índice de anormalidad. Por otra parte, será necesario introducir al sistema los valores de probabilidad de fuera de vocabulario (Poov), ya que son necesarios para obtener el valor de p_{qtot}, presente en la ecuación 3.13, en caso que una cadena de caracteres del URI que se esté analizando no se encuentre en el conjunto de observaciones "O". Además, se requiere el valor del parámetro θ para evaluar el índice de anormalidad según lo estipulado en la ecuación 3.11 .

Entonces, en conclusión, los parámetros que requieren ser introducidos al sistema de detección de intrusiones para que este funcione de manera correcta son: el conjunto de observaciones de cada estado (O), el conjunto de vectores de probabilidad de las palabras observadas en cada estado (B_S, B_P, B_A, B_V), los valores de probabilidad de fuera de vocabulario, es decir $P_{oovS}, P_{oovP}, P_{oovA}, P_{oovV}$ y el valor del parámetro θ. Todos estos parámetros serán leídos por la función de lectura del modulo de evaluación.

Una vez leídos los parámetros necesarios para que el sistema funcione, estos serán enviados a la función de evaluación junto a los segmentos de URI dados por el módulo de segmentación. Una vez recibidos los datos de entrada, esta función se encargará de hallar el índice de anormalidad (N_s) mediante la expresión 3.14 para, de este modo compararlo con el parámetro θ como se indica en 3.11.No obstante, las tareas de calculo y evaluación del índice de anormalidad, fueron subdivididas a su vez, en cuatro funciones. La primera de las funciones será la encargada de calcular tanto el ε_0 que aparece en la expresión 3.12 como la sumatoria de los logaritmos de los p_{qtot} que se encuentra en 3.14; la segunda función tomará el valor de ε_0 y la sumatoria de los logaritmos de los p_{qtot} que calculó la primera función y procederá a efectuar todas las operaciones que hay en la expresión 3.14 para de este modo obtener el índice de anormalidad, N_s; por otra parte, la tercera función tendrá como tarea comparar el índice de anormalidad N_s calculado por la tercera función, con el parámetro θ como se indica en 3.11; la cuarta será la encargada de escribir en un archivo de texto aquellos URIs anómalos.

El diagrama de bloques presente en la figura 5.4 representa el funcionamiento del módulo de evaluación.

5.3.3.3. Módulo de entrenamiento

El modelo de normalidad es uno de los aspecto importantes dentro del IDS basado en SSM ya que a partir de la información que este almacena se podrá decidir si un URI es anómalo o no dependiendo de la similitud que posea este con los datos que almacena el modelo. Por esta razón, realizar un modelo de normalidad apropiado es un aspecto importante para que el módulo de evaluación haga detecciones de intrusiones más certeras.

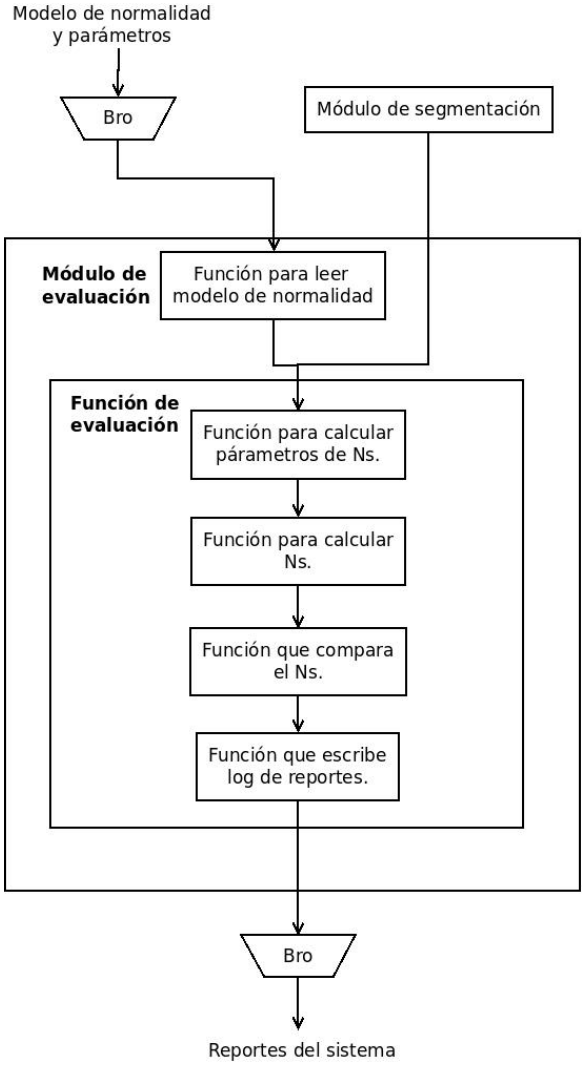


FIGURA 5.4: Diagrama de bloques del módulo de evaluación.

El módulo encargado de elaborar los modelos de normalidad será el módulo de entrenamiento. A grandes rasgos, este se encargará de recibir un conjunto de segmentos libres de ataques proporcionados por el módulo de segmentación para ir calculando la probabilidad de aparición de cada una de las palabras que aparecen en los mismos, para finalmente, escribir en un archivo toda la información recolectada, es decir, el conjunto de palabras observadas mientras se hacía el entrenamiento, sus probabilidades de aparición y el estado del autómata en el que se observaron las mismas.

Este módulo consta de dos modos: el modo "Offline" y el modo "Online". La diferencia fundamental entre el modo "Online" y el "Offline" es que en el primero necesita como entrada un modelo de normalidad ya construido y los segmentos

de URI proporcionados por el módulo de segmentación. Una vez realizado el entrenamiento, esta modalidad se encargará de actualizar el modelo que se leyó al inicio. Por otra parte, el modo "Online" solo requiere como entrada los segmentos de URI y como salida proporcionará un modelo de normalidad construido desde cero.

El modo de entrenamiento "Offline" fue dividido en tres funciones. La primera función irá observando los segmentos de los URIs de las peticiones que van llegando y contará el numero de veces que cada segmento fue observado durante el entrenamiento. La segunda función, toma el resultado de las observaciones realizadas por la primera función y calculará la probabilidad de aparición de cada uno de los segmentos haciendo uso de la formula 3.18. La tercera función tomara los resultados otorgados por la segunda función y los escribirá en un archivo de texto. Este archivo representara el modelo de normalidad construido.

Por otra parte, las tareas a realizar por el modo de entrenamiento "Online" fueron divididas de igual modo en tres funciones: La primera función se encargara de leer el modelo de normalidad; la segunda se encargará de tomar el modelo de normalidad leído por la primera función y el conjunto de segmentos de URI del módulo de segmentación. Esta información sera utilizada por la misma para ir calculando la probabilidad de aparición de los segmentos observados. La tercera función se encargara de recibir los resultados obtenidos por la segunda función y actualizara el modelo de normalidad previamente existente.

En la figura 5.5 se puede apreciar el diagrama de bloques que describe el funcionamiento del módulo de entrenamiento.

5.4. Implementación del sistema

Una vez descrita la arquitectura modular del sistema y las funcionalidades asociadas a los diferentes componentes del sistema, en este capítulo se detallarán las cuestiones más relevantes en cuanto a su implementación. Así, en primer lugar, se describirán como se implementó el filtro de paquetes HTTP/GET, para luego explicar las estructuras de datos utilizadas por cada uno de los módulos;

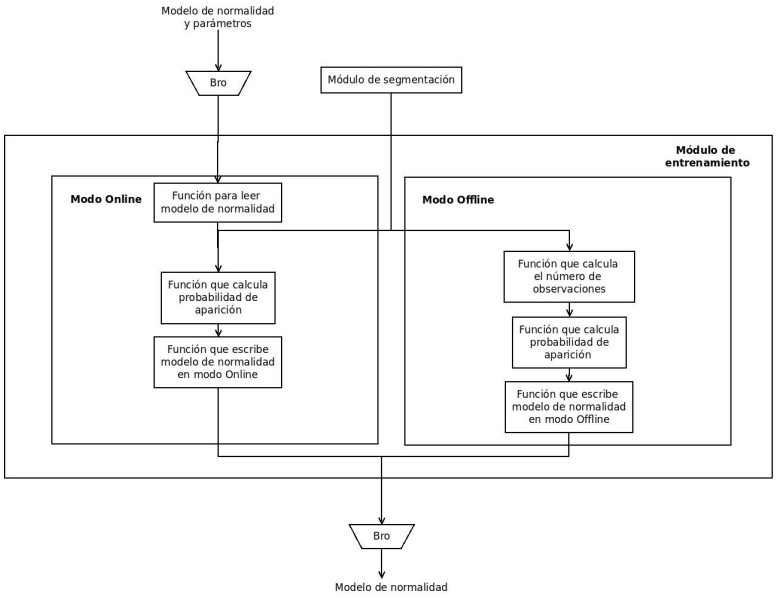


FIGURA 5.5: Diagrama de bloques del módulo de entrenamiento.

los parámetros de entrada, salida y la implementación de las funciones que los componen.

5.4.1. Filtrado de paquetes HTTP/GET

El módulo de segmentación requiere como entrada los URIs extraidos de las peticiones de tipo HTTP/GET. Esta tarea se realizó haciendo uso de las herramientas brindadas por Bro. En concreto, esta funcionalidad fué implementada con un evento primitivo de Bro llamado "http request". Este evento se genera y almacena la información de paquetes HTTP cuando son percibidos por dicha herramienta en la red. Su formato es el siguiente:

http request(c: **connection**, method: **string**, original_URI: **string**, unescaped_URI: **string**, version: **string**) donde:

- c: es una estructura de datos de tipo "connection" que almacena de manera desglosada la información del mensaje HTTP.
- method: Es el método extraído de la petición (e.g., GET,POST).
- original_URI : URI extraído de la petición (sin decodificar).

- unescaped_URI : URI extraído de la petición con todas las codificaciones de porcentaje decodificadas.
- version: Número de la version HTTP (e.g. 1.1). Este estará especificado en la petición.

5.4.2. Implementación del modulo de segmentación

En esta sección, se explicará de manera detallada la implementación del módulo de segmentación y las estructuras de datos utilizadas en el mismo. Este módulo forma parte de los tres que conforman el sistema de detección de intrusiones basado en SSM.

5.4.2.1. Estructura de datos

En el módulo de segmentación existen tres estructuras de datos fundamentales. Una es utilizada en la función de normalización y la otras dos en la de segmentación.

Normalización

La estructura de datos utilizada en la función de normalización es un hash table que contiene los elementos de un encoding table de tipo UTF-8, en donde las claves de la tabla serían los elementos de tipo UTF-8 y los atributos de las mismas corresponderían a los caracteres sin alguna codificación. Con esta tabla, se quiere implementar un diccionario que mapee los elementos de tipo UTF-8 con sus respectivos caracteres lo mas rapido posible.

Se utilizó un hash table para implementar este diccionario debido a que es ampliamente conocido que este tipo de tablas propocionan mucha eficiencia en el tiempo de búsqueda de sus elementos [25].

Para implementar este tipo de estructura se utilizó el tipo de dato “table” otorgado por el lenguaje de scripting de Bro.

Función de segmentación

Por otra parte, la estructura de datos utilizada en la función de segmentación de este módulo es un registro en el cual se almacenan todos los segmentos que se originan a partir de la segmentación de un URI, el URI sin segmentar, un booleano que informa si el URI segmentado sigue con la sintaxis establecida en el RFC 3896, y el número de estados que fueron visitados en el autómata presentado en el modelo teórico.

Se escogió un registro para almacenar la información ya que el lenguaje de "scripting" de Bro solo presenta como alternativa el tipo de dato "record" para crear un tipo de dato compuesto.

El registro creado esta conformado por los siguientes campos:

- uri: Es una variable de tipo "string" que almacena el URI sin segmentar. Este campo del registro será utilizado por el módulo de entrenamiento al momento de escribir los logs correspondientes.
- host: Es una variable de tipo "hash table" cuyas claves son número y sus valores son de tipo string. En esta sección se almacenan los segmentos del URI correspondiente al host.
- path: Es una variable de tipo "hash table" cuyas claves son número y sus valores son de tipo string. En esta sección se almacenan los segmentos del URI correspondiente al path.
- query: Es una variable de tipo "hash table" cuyas claves y sus valores son de tipo string. En esta sección se almacenan los atributos y los valores del query del URI en caso de que el mismo posea. Los atributos y los valores se almacenarán en una hash table como se mencionó con anterioridad, en donde la clave del mismo serán los atributos del query y en donde los valores de las misma serán los valores del query del URI.
- fragment: Es una variable de tipo "string" que almacenará el fragment del URI en caso de poseerlo.
- número de estados: es una variable de tipo entero que almacena el número de estados del autómata que se emplea en el módelo teórico fueron visitados

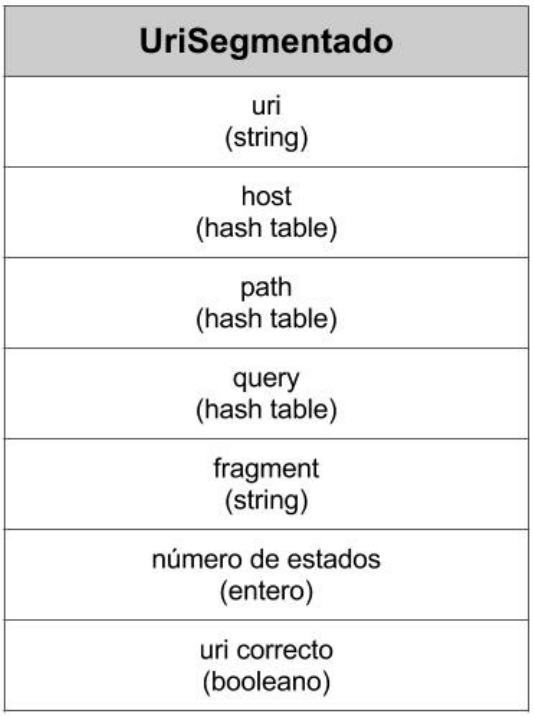


FIGURA 5.6: Ejemplo gráfico de la estructura "UriSegmentado".

para reconocer el URI. Este campo del registro será utilizado por el módulo de evaluación.

- uri correcto: es una variable de tipo booleano que indica si el URI está sintácticamente correcto o no.

El nombre que recibirá este tipo de estructura es "UriSegmentado". La misma se puede apreciar gráficamente en la figura 5.6.

Además de utilizar el tipo de dato "UriSegmentado", el módulo de segmentación hace uso de una estructura de datos compuesta, primitiva del lenguaje de "scripting" de Bro llamada "URI". Dicha estructura posee los siguientes campos:

- scheme: Es un campo opcional de tipo "string" que almacena el protocolo del URI.

- netlocation: Es un campo de tipo "string" que almacena el nombre de dominio o la dirección IP del URI.
- portnum: Es un campo opcional de tipo "count" que almacena el número de puerto del URI.
- path:Es un campo de tipo "string" que almacena la ruta del URI.
- file_name: En un campo opcional de tipo "string" que almacena el nombre completo del archivo de la ruta del URI junto a su extensión.
- file_base: En un campo opcional de tipo "string" que almacena el nombre del archivo de la ruta del URI sin su extensión.
- file_ext: En un campo opcional de tipo "string" que almacena la extensión del archivo de la ruta.
- params: Es un campo opcional de tipo "table" que almacena todos los parámetros de la consulta del URI. Esta tabla mapea todos los atributos con sus valores.

La estructura "URI" se puede apreciar gráficamente en la figura 5.7.

5.4.2.2. Función de segmentación

En la función de segmentación, el problema que se quiere resolver es el siguiente:construir un analizador sintáctico que haga uso de una gramática (libre de contexto) de atributos que genera el mismo lenguaje que reconoce el autómata presentado en la figura 2.4. A nivel de implementación lo que se quiere construir es un analizador sintáctico que verifique que los URIs tengan una estructura sintáctica correcta según los estándares del RFC 3986, y que a la vez vaya almacenando los segmentos, delimitados por los delimitadores presentados en la sección 2.6, en la estructura de datos de la figura 4.2, detallada en la sección 4.2.1.

A continuación se presentara la forma en que se modeló y se implementó el problema presentado con anterioridad haciendo uso de las herramientas presentadas por el lenguaje de "scripting" de Bro.

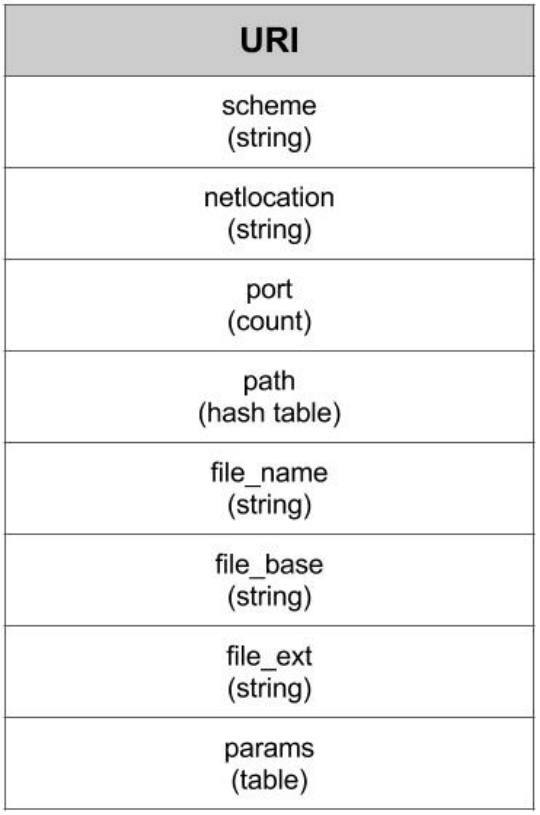


FIGURA 5.7: Ejemplo gráfico de la estructura "URI".

Para realizar el analizador sintáctico se requiere construir una gramática libre de contexto.

Antes de presentar la gramática se mostrarán los tokens que utilizará la misma como elementos terminales.

Estos tokens utilizarán las expresiones regulares que soporta Bro. Estas están basadas en las expresiones regulares empleadas en Lex, la librería para realizar análisis léxicos del lenguaje de programación C. Estos tokens utilizarán las expresiones regulares que soporta Bro. Estas están basadas en las expresiones regulares empleadas en Lex, la librería para realizar análisis léxicos del lenguaje de programación C. La lista de tokens utiliza es la siguiente:

- protocolo: "http"—"https"

- host: $(([a-z]+[a-z0-9]*[.])?([a-z0-9]+[a-z0-9]*[.])+[a-z]2,3|localhost)(:([0-9]1,5))?|$
  $((25[0\text{-}5]—2[0\text{-}4][0\text{-}9]—[01]?[0\text{-}9][0\text{-}9]?)\dot{)}3(25[0\text{-}5]—2[0\text{-}4][0\text{-}9]—[01]?[0\text{-}9][0\text{-}9]?)(:([0\text{-}9]1,5))?$
- elementoPath: $[\hat{?}\#]*$
- atributo: $[\hat{\#}\&=]$
- valor: $[\hat{\#}\&]$
- fragmento: $(\#.*)$

Por otra parte, la gramática libre de contexto en la que está basada la función de análisis sintáctico es la siguiente.

$$\begin{aligned}
S &-> protocolo\backslash :\ //"H \\
H &-> host\backslash /"P|host \\
P &-> P'\backslash ?"Q|P'\backslash \#"F|P' \\
Q &-> Q'\backslash \#\backslash F|Q' \\
F &-> fragment \\
P' &-> \lambda \\
P' &-> elementoPath \\
P' &-> elementoPath\backslash /"P' \\
Q' &-> atributo\backslash =" valor \\
Q' &-> atributo\backslash =" valor\backslash \&"Q'
\end{aligned} \tag{5.1}$$

Donde, el elemento de inicio de esta gramática es el símbolo no terminal “S”

Para almacenar los segmentos en la estructura de datos compuesta de tipo “uriSegmentado” que se presentó en la sección 5.4.2.1 a la gramática 5.1 se le van a asociar atributos a algunas de las reglas de la misma. A este tipo de gramática se le llama gramática de atributos. Los atributos de dicha gramática serán escritos en el lenguaje de scripting de Bro.

$$
\begin{aligned}
S->protocolo\backslash://"Huri\$protocolo := protocolo \\
H->host\backslash/"P|hosturi\$host := host \\
P->P'\backslash?"Q|P'\backslash\#"F|P' \\
Q->Q'\backslash\#\backslash F|Q' \\
F->fragmenturi\$fragment := fragment \\
P'->\lambda uri\$path.append(\backslash") \\
P'->elementoPathuri\$path.append(elementoPath) \\
P'->elementoPath\backslash/"P'uri\$path.append(elementoPath) \\
Q'->atributo\backslash="valoruri\$query[atributo] := valor \\
Q'->atributo\backslash="valor\backslash\&"Q'uri\$query[atributo] := valor
\end{aligned}
\tag{5.2}
$$

La gramática de atributos presentada 5.2 representa el modelo de lo que se implementó haciendo uso de Bro.

Como Bro no cuenta con una librería propia para programar un parser de manera sencilla, se hizo uso de las herramientas con las que cuenta este para procesar cadena de caracteres y expresiones regulares. Las funciones fundamentales que se utilizaron para implementar la gramática 5.2 fueron: "split", "split_all" y "decompose_uri".

A continuación se explicara un poco el funcionamiento de cada una de ellas.

- split: La función "split" tiene la siguiente forma:

  split(str: **string**, re: **pattern**) : **string_array Attributes:&deprecated**

  Esta función se encarga de dividir una cadena de caracteres de acuerdo a un patrón e introduce el resultado en un arreglo. Por ejemplo, $split"a-b--cd", /() + /)$ retorna $\{"a","b","cd"\}$.

  Los parámetros de dicha función son:

  - Str:La cadena de caracteres que se quiere dividir.
  - Re: El patrón que describe lo delimitadores mediante los cuales será dividida la cadena de caracteres.

- Retorna: Un arreglo de caracteres donde cada elemento corresponde a una subcadena de caracteres de Str separado por Re.

- split_all(): Esta función realiza el mismo trabajo que "split" con la diferencia que los separadores son incluidos también el parámetro de salida. Por ejemplo, $split\_all("a-b--cd", /() + /)$ retorna $\{"a", "-", "b", "--", "cd"\}$.

- decompose_uri: La función decompose_uri tiene la siguiente forma:

  decompose_uri(uri: **string**) : **URI**

  decompose_uri dado un URI, retorna una estructura de datos compuesta de tipo URI explicada en la sección 5.4.2.1 que contiene información como el protocolo,la ruta, el número de puerto y los parámetros de las consultas del URI recibido.

Por otra parte, para implementar la gramática 5.2 se modelaron los elementos no finales de la misma como funciones. Por otra parte, los elementos finales, fueron extraídos haciendo uso de "split" y "split_all".

En conclusión, la función encargada de segmentar y realizar el análisis sintáctico del módulo de segmentación posee la siguiente estructura:

segmentar(uri: **string**) : **UriSegmentado**. La función se encargará, de tomar una cadena de caracteres y segmentarla según delimitados por los delimitadores presentados en la sección 3.6 y almacenar los mismo en la estructura "UriSegmentado". Los parámetros de dicha función son:

- url : Cadena de caracteres que será segmentada.

5.4.2.3. Función de normalización

La normalización de los URI debe realizarse de acuerdo a las especificaciones establecidas en la Sección 2.6, habiéndose implementado para ello una función denominada "normalizar", con el siguiente formato.

normalizar(url: **string**): **string** Donde:

- url: Variable de tipo "string" que sera normalizada.

Esta función tomará el parámetro de entrada y mediante un "loop for" se itera por cada una de las claves de la tabla que se presentó en la sección 5.4.2.1. Si alguno de los elementos de la tabla está contenido en el "string" que la función recibe como entrada, entonces se procederá a reemplazar el elemento del párametro de entrada por el valor que posee dicha clave en la "hash table". Esta tarea de subsitución se realizará haciendo uso de la función "subst string" que proporciona Bro.

"subst_string" es una función de la forma:

subst_string(s: **string**, from: **string**, to: **string**) : **string** Se encarga de hacer sustituciones en una cadena de caracteres. Los parámetros de esta son:

- S:Cadena de caracteres en la que se efectúa la sustitución.
- From: La cadena de caracteres que se va a buscar en "S" para ser sustituida.
- To: Cadena de caracteres que pasará a sustituir a "From".

5.4.3. Implementación del módulo de evaluación

En esta sección se explicará de manera detallada las estructuras de datos utilizadas y la manera en la que se implementó el módulo de evaluación explicado en la sección 5.3.3.2 haciendo uso del lenguaje de "scripting" de Bro.

5.4.3.1. Estructura de datos

El módulo de evaluación del sistema cuenta con varias estructura de datos compuestas de tipo "register" y de tipo "table" que serán explicados a continuación

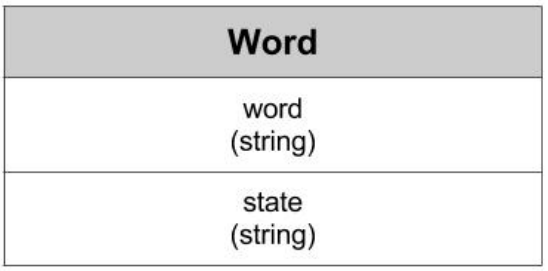


FIGURA 5.8: Ejemplo gráfico de la estructura "Word".

Modelo de normalidad

Existen dos estructuras de tipo "register" en la función para leer el modelo de normalidad y dos de tipo "table". Las estructuras de tipo "register" poseen los siguientes campo:

La primera, cuyo nombre es "Word" posee los siguientes campo:

- "word" es un campo de tipo "string" en donde se almacenarán las palabras del vocabulario de cada uno de los estados del autómata presentado en la imagen 3.8.
- "state" es un campo de tipo string en donde se indicará el estado del autómata presentado en la imagen 3.8 al que pertenece la palabra almacenada en el campo "word".

Esta estructura se puede observar de manera gráfica en la figura 5.8.

La segunda estructura llamada "Probability" solo posee un campo llamado "probability".

- "probability" es un campo de tipo "double" en donde se almacenará la probabilidad de generación de las palabras que se encuentran en el modelo.

Se puede observar un ejemplo gráfico de la estructura en la figura 5.9.

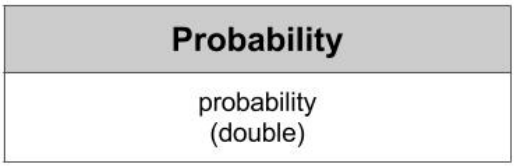


FIGURA 5.9: Ejemplo gráfico de la estructura "Probability".

La tercera estructura de tipo "register" que será utilizada en el módulo de evaluación se llamada "TableDescription" y pertenece al módulo "Input" de Bro y se utilizará al momento de leer los archivos de entrada. Los campos utilizados de esta estructura fueron los siguientes:

- source: campo de tipo "string" que almacena el nombre del archivo que va a ser leído.
- name: campo de tipo "string" que almacenará el nombre que se le asignará al flujo de entrada.
- destination: Nombre de la tabla que almacena la información contenida en los archivos.
- idx: Nombre del registro que definirá los valores que utilizará la tabla que almacena la información del archivo como clave.
- val: Campo opcional que almacena el nombre del registro que define los valores de la tabla que almacena la información de los archivos de entrada.

En la figura 5.10 se muestra un ejemplo gráfico de "TableDescription".

Por otra parte, las estructuras de tipo "table" son dos. Una tabla será utilizada para almacenar las palabras del vocabulario, la probabilidad de generación y el nombre del estado al que pertenece dicha información mientras que la otra almacenará las probabilidades de fuera de vocabulario de cada uno de los estados y el valor del parámetro θ presente en la expresión 3.11.

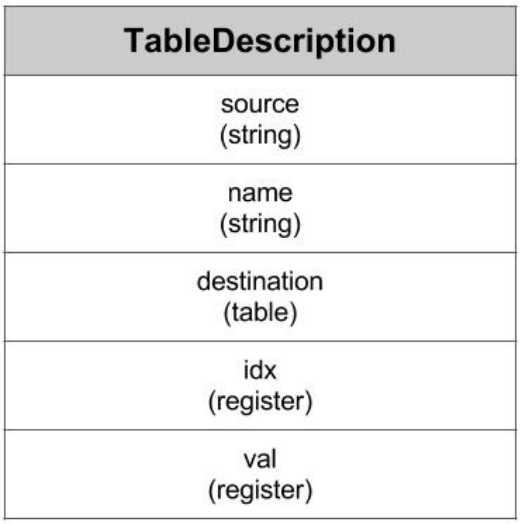


FIGURA 5.10: Ejemplo gráfico de la estructura "TableDescription".

La primera tabla es una "hash table" que posee dos elementos tipo "string" como clave, y como valor tiene un campo de tipo "Probability".

Esta tabla almacenará el modelo de normalidad del sistema, es decir, las palabras del vocabulario, la probabilidad de generación de las mismas y el estado al que pertenece dicha información. Las claves de esta tabla serán las palabras del vocabulario y el nombre del estado al que pertenecen. Toda esta informacion será dada a través de un archivo de texto cuya estructura será explicada en las sección 5.4.3.2.

La segunda tabla es una hash table que tendrá solo un elemento como clave de tipo "string". Los valores de la misma serán campo tipo "Valor".

En dicha tabla se van a almacenar las probabilidades de fuera de vocabulario de cada uno de los estados del autómata y el parámetro θ. La clave de la misma sería una cadena de caracteres que identifique cada una de las probabilidades y el parámetro θ. Esas etiquetas serían las siguientes: Poov1, Poov2, Poov3, Poov4, Theta.

Evaluación de las probabilidades de los URI

La función que se encarga de calcular el índice de anormalidad y evaluar si el mismo es anómalo o no, solo contiene una estructura compuesta de tipo "record" y se utiliza junto a una herramienta de Bro que funciona para escribir "logs". Los "logs" que son escritos a través de esta herramienta son archivos de texto que contiene una lista de los URI que presentan anomalías.

La estructura lleva como nombre "InfoAtaque" y contiene los siguientes campos:

- clasificacion: es un campo de tipo "string" que almacena la clasificación de los URIs anómalos, es decir, aquí se indica si el mismo presenta anomalía por estar sintácticamente mal construidos o porque el índice de anormalidad sobrepasó el parámetro θ.
- uri: es un campo de tipo "string" en el que se va a almacenar el uri que va a ser registrado en el log.
- probability: es un campo de tipo "string" en donde se va a almacenar el valor del índice de anormalidad del uri.

Todos los campos anteriormente descritos poseen la cualidad de ser de tipo "log" tambien. Con esto se indica cuál de los elementos de la estructura de datos se escribirá sobre los logs.

El registro "InfoAtaque" se puede observar de manera gráfica en la figura 5.11.

5.4.3.2. Modelo de normalidad en Bro

En esta sección se explicará cual es el formato de almacenamiento del modelo de normalidad y los parámetros de configuración, y se explicará como se implementó la lectura del mismo.

Los elementos que necesitan ser introducidos al módulo de evaluación para su correcto funcionamiento son: el conjunto de observaciones de cada estado

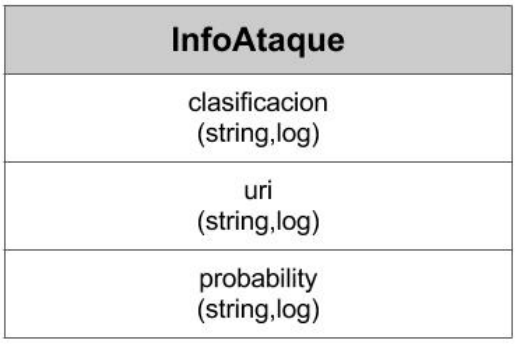


FIGURA 5.11: Ejemplo gráfico de la estructura "InfoAtaque".

(O), el conjunto de vectores de probabilidad de las palabras observadas en cada estado (B_S, B_P, B_A, B_V), los valores de probabilidad de fuera de vocabulario ($P_{oovP}, P_{oovA}, P_{oovV}$), y el valor del parámetro θ (Ec. 3.11).

Formato de los archivos de entrada

Los parámetros necesarios para el funcionamiento del módulo de evaluación, serán introducidos a tráves de dos archivos, un archivo llamado "config" que contendrá los valores de la probabilidad de fuera de vocabulario ($P_{oovP}, P_{oovA}, P_{oovV}$) y el valor del parámetro θ (ec. 3.11) y otro llamado "modeloBro.log" en el cual estará el conjunto de observaciones de cada estado junto a sus probabilidades de generación.

Tanto el archivo "config" como el "modeloBro.log" están regidos bajo un formato que establece la herramienta de Bro para leer archivos de entrada.

El formato que establece Bro para los archivos de entrada define que la información se debe introducir en columnas separada mediante tabs, y que debe existir un encabezado al inicio del mismo que inicie abriendo con un numeral (#) seguido de la palabra "fields". Luego de esta palabra se escribirán los nombres que se le asignará a cada columna del archivo.

```
#fields    clave    valor
Poov1    0.0001
Poov2    0.0001
Poov3    0.0001
Poov4    0.0001
Theta    12.0
```

FIGURA 5.12: Formato de archivo "config".

Por lo tanto, el formato del archivo "config" sería como el que se muestra en la figura 5.12.

La primera columna corresponde a las etiquetas que identifican los valores que se encuentran en la segunda columna. Es importante recalcar que los nombres "Poov1", "Poov2", "Poov3", "Poov4" y "Theta" deben ser escritos en el archivo obligatoriamente de la misma forma en la que aparecen en el ejemplo.

Por otro lado, el formato del archivo "modeloBro.log" es el que se muestra en la figura 5.13.

La segunda columna de este archivo ("word") corresponde al conjunto de observaciones "O" de cada uno de los estados, mientras que la tercera columna ("probability") indica la probabilidad de aparición de cada palabra (B_S, B_P, B_A, B_V). Por otra parte, la primera columna ("state") indica a que estado del autómata 3.8 pertenece cada palabra y su probabilidad de generación.

Las filas del archivo "modeloBro.log" que tienen en la primera columna las palabras: "numeroPalabraSs", "numeroPalabraSp", "numeroPalabraSa", "numeroPalabraSv" contienen el número de palabras que han sido procesadas por estado. Esta informacion será utilizada en el entrenamiento en modo offline que será explicado en la sección **??**.

```
#fields    state    word    probability
#types    string    string    double
Bss    192.168.1.13    1.0
Bsp    idr    0.16759
Bsa    idpd    0.000006
Bsv    nombre    0.000175
numeroPalabraSs    numTotal    205069.0
numeroPalabraSp    numTotal    423870.0
numeroPalabraSa    numTotal    319727.0
numeroPalabraSv    numTotal    319727.0
```

FIGURA 5.13: Formato de archivo "modeloBro.log".

Lectura de los archivos

Una vez explicado el formato de los archivos que contienen los elementos del modelo y los parámetros necesarios para realizar el cálculo y la evaluación del índice de normalidad de los URIs a evaluar se procederá a explicar el funcionamiento de las funciones encargadas de leer los archivos con la información de entrada.

Para leer dicha informacion se hizo uso del "Input Framework" que otorga Bro como herramienta para leer archivos de entrada. La función utilizada para leer los archivos se llama "add_table" y pertenece al módulo llamado "Input".

"add_table" se encarga de leer archivos de entrada y almacenar sus datos en una tabla. El formato de esta funcion es el siguiente:
Input::add_table(description: **Input::TableDescription**) : **bool**

Donde "**TableDescription**" es la estructura descrita graficamente en la figura 5.10.

5.4.3.3. Evaluación de las probabilidades de los URI

La función "evaluacion" del módulo de evaluación, se encargará de hallar el índice de anormalidad, dado un conjunto de segmentos de URI, e indicará si el mismo es anómalo o no comparándolo con el valor de θ, para luego escribir los resultados de la evaluación en un archivo de reportes del sistema. La función implementada que se encargara de realizar las tareas anteriormente descritas tiene el siguiente formato:

evaluacion(uriParsed: **UriSegmentado**, Bvector: **table[string,string] of Probability**, config: **table[string] of Valor**) donde:

- uriParsed: es un parámetro de tipo "UriSegmentado" (figura 5.6), que almacena los segmentos del URI.
- Bvector: es una tabla que almacena el conjunto de observaciones de cada estado (O) y el conjunto de vectores de probabilidad de las palabras observadas en cada estado, es decir, B_S, B_P, B_A, B_V.
- config: Es una tabla que almacena los diferentes Poov y el valor de θ.

No obstante, la tarea realizada por "evaluacion" fue subdividida en cuatro funciones: "epsiloSumatoria", "calcularIndiceAnormalidad", "evaluarIndiceAnormalidad" y "escribirReporte". La función "epsiloSumatoria" se encargará de calcular tanto el ε_0 que aparece en la expresión 3.12 como la sumatoria de los logaritmos de los p_{qtot} que se encuentra en la expresión 3.14.

El formato de dicha función es el siguiente:

epsiloSumatoria(segmentos: **UriSegmentado**, Bvector: **table[string,string] of Probability**, epsilon : **double**, estado:**string**): **table[count] of double**

donde los parámetros de entrada serán:

- segmentos: es un parámetro de tipo "UriSegmentado" (figura 5.6), que almacena los segmentos del URI.

- Bvector: es una tabla que almacena el conjunto de observaciones de cada estado (O) y el conjunto de vectores de probabilidad de las palabras observadas en cada estado, es decir, B_S, B_P, B_A, B_V.
- epsilon: Valor correspondiente al Poov.
- estado: estado del autómata que se esta evaluando (“host”,“path”,“argumentos”, “valores”).

El parámetro de salida correspondería a una tabla que contiene dos elementos: el resultado de la suma de las probabilidades de aparición de las palabras y la suma del los logaritmos de la probabilidad de aparición.

Por otra parte, “calcularIndiceAnormalidad”, la encargada de calcular el índice de anormalidad N_s , tiene el siguiente formato:

calcularIndiceAnormalidad(epsilon0: **double**, N: **double**, sumaLogaritmos: **double**) : double. Donde:

- epsilon0: Valor de ε_0 (expresión 3.12).
- N: Valor de “T” en la expresión (expresión 3.12).
- sumaLogaritmos: Suma de los logaritmos de p_{qtot} que se encuentra en la expresión (expresión 3.12).

El parámetro de salida de esta función será el valor del índice de anormalidad.

La función, “evaluarIndiceAnormalidad” que tiene como tarea comparar el índice de anormalidad (N_s) con el parámetro θ, de la forma en que se muestra en la expresión 3.11 tiene el siguiente formato:

evaluarIndiceAnormalidad(theta: **double**,indiceAnormalidad: **double**)

donde:

- theta: Umbral de normalidad (θ).

- indicesAnormalidad: Índice de anormalidad.

La ultima función, "escribirReporte", la encargada de escribir en el archivo de reportes la información de los URIs que se han detectado como anómalos presenta el siguiente formato:

escribirReporte(clasificacion: **string**, uri: **string**,indiceAnormalidad: **string**)

- clasificacion: En este campo se informa si el URI es anómalo por sobrepasar el umbral de normalidad o por estar construido de manera incorrecta sintácticamente.
- uri: En este campo se introduce el URI que se quiere escribir en el archivo de salida.
- indiceAnormalidad: Este campo corresponde al índice de anormalidad.

La función anterior fue implementada con la ayuda de las herramientas que aporta el "Logging Framework" de Bro y la estructura de datos "InfoAtaque" (fig. 5.11). La funciones del "Framework" utilizadas fueron: "write" para escribir sobre el archivo y "create_stream" para crearlo. Ambas pertenecen al modulo "LOG".

El formato que sigue la función "write" es el siguiente :

LOG::InfoAtaque(id: **Log::ID**, columns: **any**) : **bool** Sus parámetros de entrada son:

- id: Es al ID asociado al archivo sobre el cual se va a escribir.
- column: Registro que contiene los valores que van a ser escritos en el archivo.

El parámetro de salida de esta función es "verdadero" si el archivo fue encontrado y no existieron problemas al momento de la escritura, y "falso" de lo contrario.

Por otra parte, la función "create_stream", esta definida de la siguiente forma:

LOG::create stream(id: **Log::ID**, stream: **Log::Stream**) : **bool**

Los parámetros de entrada de la misma son:

- id: Es al ID asociado al archivo sobre el cual se va a escribir.
- stream: Registro que almacenará la ruta en donde se quiere crear el archivo y la estructura de dato que se utilizará en la escritura del archivo.

El parámetro de salida de esta función sera "verdadero" si se puedo crear el archivo y "falso" si es posible.

5.4.4. Implementación del módulo de entrenamiento

Como se pudo apreciar en la sección 5.3.3.3 el modo de entrenamiento cuenta con dos modos: el modo "Online" y el modo "Offline". En esta se sección, se explicarán las estructuras de datos utilizadas por ambos modos y las funciones implementadas en cada uno de ellos.

5.4.4.1. Estructura de datos

El módulo de entrenamiento consta de varias estructuras de datos de tipo "register" y de tipo "table" para realizar su trabajo. Tanto el modo "Online" como el "Offline" comparten las mismas estructuras. La estructuras serían las siguientes: "Entrenamiento" es una estructura de tipo "record", cuyo funcionamiento es ir almacenando el número de veces que una palabra es observada junto con la probabilidad de observación, mientras se realiza el entrenamiento. Los campos de esta estructura son los siguientes:

- numPalabras: campo de tipo entero que almacena el número de veces que una palabra es observada durante el entrenamiento.
- probability: campo de tipo flotante que almacene la probabilidad de observación de una palabra.

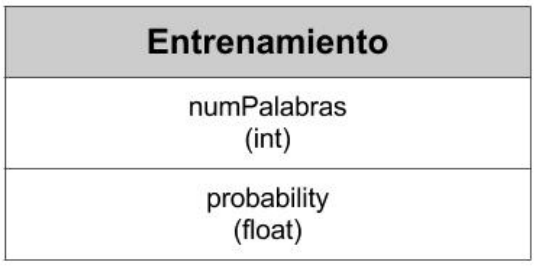


FIGURA 5.14: Ejemplo gráfico de la estructura "Entrenamiento".

Un ejemplo gráfico de la estructura "Entrenamiento" se puede apreciar en la figura 5.14. La otra estructura de tipo "record" es "Info". "Info" se encargará de almacenar la información que será escrita en el archivo de texto que representará el modelo de normalidad. Los campos de esta estructura de datos son los siguientes:

- state: es un campo de tipo "string" que almacenará el nombre del estado al que pertenece la palabra y su probabilidad de observación.
- word: es un campo de tipo "string" que almacenará las palabras observadas durante el entrenamiento.
- probability: es un campo de tipo flotante que almacenará la probabilidad de observación de las palabras.

La estructura "Info" se puede observar de manera gráfica en la figura 5.15.

Por otra parte, la estructura tipo "table" que se utilizó en la implementación de este módulo fueron las siguientes: Se hizo uso de una tabla de hash que posee como clave un campo tipo "string" y como valor una estructura de datos de tipo "Entrenamiento" (fig. 5.14). Esta tabla tendrá como función almacenar las palabras que van apareciendo durante el entrenamiento, el número de veces que fueron observadas las mismas y la probabilidad de aparición . La clave de esta tabla serán las palabras observadas y el resto de la informacion sería almacenado en la estructura de datos "Entrenamiento". Cada estado del autómata tendrá su tabla de entrenamiento propia para de esta manera tener una mayor organización de las observaciones realizadas durante el entrenamiento.

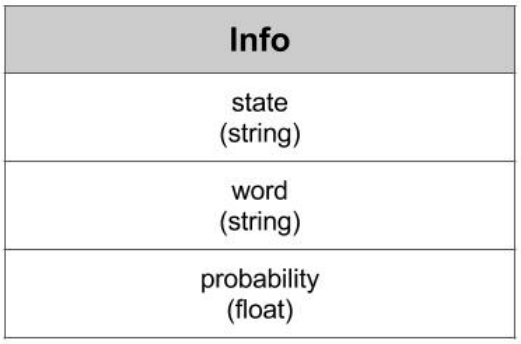


FIGURA 5.15: Ejemplo gráfico de la estructura "Entrenamiento".

5.4.4.2. Entrenamiento "Offline"

El modo de entrenamiento "Offline", es el encargado de crear un modelo de normalidad desde cero.Esta modalidad está conformada por una función general llamada "entrenarOffline", que fué subdividida en tres funciones: "entrenamientoOffline", "evaluarProbabilidad" y "escribirArchivoOffline".

"entrenarOffline", será la función encargada de llamar el resto de las funciones, necesarias para realizar el entrenamiento en modo "Offline". Su formato es el siguiente:

entrenarOffline(segmentos: **UriSegmentado**) donde:

- segmentos: campo de tipo "UriSegmentado" (fig. 5.6), que almacena los segmentos de un URI

Por otra parte, la función "entrenamientoOffline" irá observando los segmentos de los URIs de las peticiones que van llegando y contará el número de veces que cada segmento fue observado durante el entrenamiento. El formato de la misma sigue la siguiente estructura:

entrenamientoOffline(segmentos: **UriSegmentado**, vocabulario: **table[string] of Entrenamiento**, estado: **string**) donde:

- segmentos: campo de tipo "UriSegmentado" (fig. 5.6), que almacena los segmentos de un URI (las observaciones).
- vocabulario: Tabla que contiene una lista de palabras y una lista de nÚmeros que corresponde al número de apariciones y probabilidad de aparición de las misma (fig. 5.14).
- numPalabras: Número total de apariciones de todas las palabras vistas durante el entrenamiento de un estado en concreto.
- estado: estado del autómata que se esta entrenando ("host", "path", "argumentos","valores").

La "evaluarProbabilidad" , encargada de tomar el resultado de las observaciones realizadas por "entrenamientoOffline" y calcular la probabilidad de aparición de cada uno de los segmentos (ec. 3.18) tiene el siguiente formato:

evaluarProbabilidad(vocabulario: **table[string] of Entrenamiento**, numPalabras: **double**)

- vocabulario: Tabla que contiene una lista de palabras y una lista de números que corresponde al número de apariciones y probabilidad de aparición de las misma (fig. 5.15).
- numPalabras: Número total de apariciones de todas las palabras vistas durante el entrenamiento de un estado en concreto.

Finalmente, "escribirArchivoOffline" toma los resultados otorgados por "evaluarProbabilidad" y los escribirá en un archivo de texto. El formato de esta función es el siguiente:

escribirArchivoOffline(vocabulario: **table[string] of Entrenamiento**, estado: **string**) donde:

- vocabulario: Tabla que contiene una lista de palabras y una lista de números que corresponde al número de apariciones y probabilidad de aparición de las misma (fig. 5.15).
- estado: estado al que pertenece la tabla "vocabulario".

5.4.4.3. Entrenamiento "Online"

El modo de entrenamiento "Online", encargado de hacer modificaciones a un modelo de normalidad previamente existente fue implementado haciendo uso de una función llamada "entrenarOnline" que a su vez esta subdividida en tres funciones. Los nombres que se les dio a estas al momento de ser implementadas fueron: "entrenamientoOnline","escribirArchivoOnline".

La función "entrenarOnline", cuya tarea sera llamar al resto de las funciones del entrenamiento en modo "Online" tiene el siguiente formato:

entrenarOnline(uriParsed:**segmentos::UriSegmentado**) donde:

- segmentos: campo de tipo "UriSegmentado" (fig. 5.6), que almacena los segmentos de un URI

Por otra parte, "entrenamientoOnline", encargada de calcular la probabilidad de aparición de los segmentos observados haciendo uso de de los segmentos de URI dado por "entrenarOnline" y el modelo de normalidad, leído por el módulo de lectura del sistema, sigue el siguiente formato:

entrenamientoOnline(segmentos: **UriSegmentado**, modelo: **table[string,string] of Probability**, numPalabras: **double**, state: **string**) donde:

- segmentos: campo de tipo "UriSegmentado" (fig. 5.6), que almacena los segmentos de un URI.
- modelo: tabla que almacena el modelo de normalidad previamente leído.
- numPalabras:
- state: nombre del estado al que se le esta realizando el entrenamiento ("host", "path","atributos","valores".

Finalmente, "escribirArchivoOnline", se encargará de recibir todos los resultados obtenidos del entrenamiento para escribirlos sobre un archivo de texto. El formato de esta función es:

escribirArchivoOnline(vocabulario: **table[string,string] of Probability**) donde:

- vocabulario: Es una tabla que almacena los resultados del entrena- miento.

5.5. Evaluación y pruebas

En el presente capítulo se explicará las pruebas funcionales y operativas realizadas al sistema. Así como las bases de datos utilizadas para realizar las mismas.

5.5.1. Base de datos

Las bases de datos utilizadas por las pruebas fueron trazas capturadas en servidores web. Para capturar estos paquetes de tipo HTTP los pasos a seguir fueron los siguientes:

1. Se instaló una aplicación de servicio web en el ordenador.
2. Se corrió una aplicación web en la aplicación anteriormente instalada.
3. Se empezaron a hacer solicitudes a la aplicación web a tráves de otro dispositivo. Mientras esto ocurria, "Wireshark" realizaba las capturas de los paquetes de la red.

Se obtuvo cuatro bases de datos:

- db1.pcap: Capturas de paquetes provenientes de las solicitudes reali- zadas a una aplicación web, A.
- db2.pcap: Capturas de paquetes provenientes de la misma aplicación web A.
- db3.pcap: Capturas de paquetes provenientes de una aplicación web, B.
- db4.pcap: Capturas de paquetes descargadas de: [26].

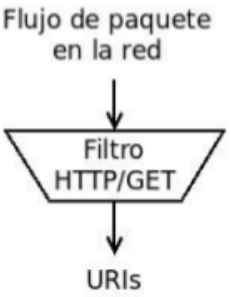


FIGURA 5.16: Esquema de pruebas realizadas al filtro HTTP/GET.

5.5.2. Pruebas funcionales

En esta sección se explicarán las pruebas funcionales aplicadas a cada una de la funciones construidas en el sistema.

5.5.2.1. Filtro HTTP/GET

Para probar el filtro de tipo HTTP/GET se tomó la base de datos, **db1.pcap**, que contiene tanto paquetes HTTP/GET, como paquetes correspondientes a la sesión HTTP. La prueba consistió en pasarle a la función construida la base de datos de paquetes para que esta imprimiera los URIs de las peticiones HTTP/GET que estuviesen contenidas en la misma. En la figura 5.16 se muestra un esquema de la prueba que se realizó.

5.5.2.2. Lectura de archivos

La lectura de los archivos es de suma importancia, ya que el sistema utiliza un modelo de normalidad y un conjunto de parámetros que se introducen al mismo a tráves de archivos. La manera en la que se probó esta función consistió en leer ambos archivos de textos necesarios por el sistema: “config” y “modeloBro.log” y luego se imprimieron las tablas resultantes de la lectura de los mismos. En la figura 5.17, se puede observar el archivo “config” que leerá la función de lectura del sistema y en la figura 5.18, se puede apreciar la forma en la que la misma almacenó los datos en la tabla correspondiente. Por otra parte, en la figura 5.19,

Clave	Valor
Poov1	1E-6
Poov2	1E-6
Poov3	1E-6
Poov4	1E-6
Theta	13.0

FIGURA 5.17: Archivo "config".

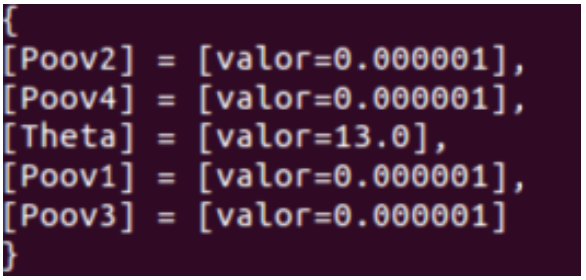


FIGURA 5.18: Tabla que contiene información del archivo "config".

Estado	Palabra	Probabilidad
Bss	192.168.3.48	1.0
Bsp	tr-tema-16.pdf	0.25
Bsp	html	0.25
Bsp	/	0.25
Bsp	tr-problemas1-14.pdf	0.25
numeroPalabraSv	numTotal	0.0
numeroPalabraSs	numTotal	4.0
numeroPalabraSp	numTotal	4.0
numeroPalabraSa	numTotal	0.0

FIGURA 5.19: Archivo "modeloBro.log".

está el archivo "modeloBro.log" que fué utilizado en la prueba realizada, mientras que en la figura 5.20 se muestra el resultado arrojado por la misma.

5.5.2.3. Módulo de segmentación

Como se mencionó en los secciones anteriores, el modulo de segmentación cuenta con dos funciones principales: la función de normalización y la de segmentación. A continuación se mostrará la manera en la que se probó cada una de las funciones.

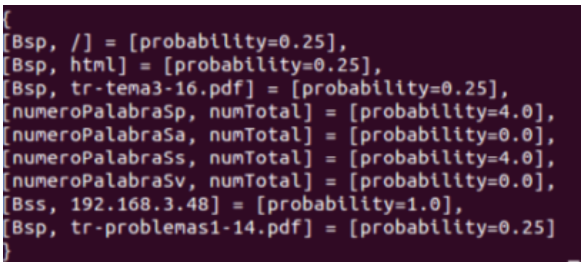


FIGURA 5.20: Tabla que contiene información del archivo "modeloBro.log".

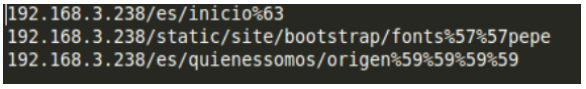


FIGURA 5.21: URIs sin normalizar.

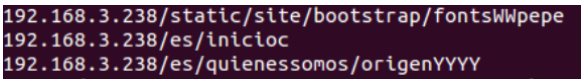


FIGURA 5.22: URIs normalizados.

Función de normalización

Las pruebas realizadas a la función de normalización consistió en realizar una lista de URIs sin normalizar, ingresarlos como parámetros a la función y observar los resultados arrojados por la misma. La lista de URIs utilizados en esta prueba se pueden observar en la figura 5.21. Por otra parte, los resultados arrojados por la misma se muestran en la figura 5.22.

Función de segmentación

La pruebas realizadas a la función de segmentación, consistieron en ingresarle un conjunto de URIs a la misma y observar la manera en la que esta los segmentaba.

5.5.2.4. Módulo de entrenamiento

Modo "Offline"

Como se pudo observar en los secciones anteriores, el módulo de entrenamiento esta conformado por una función, "entrenarOffline", que que se encarga de hacer la llamada a función de: "entrenamientoOffline", "evaluarProbabilidad", "escribirArchivoOffline". Para probar la función, "entrenamientoOffline", se construyeron una serie de datos de tipo "UriSegmentado" (fig. 5.6), provenientes de la base de datos **db3.pcap** y le fueron pasados como parámetro de entrada a la misma. Una vez procesados los datos, se observó la salida de la misma para verificar la correctitud.

Por otra parte, para probar la función "evaluarProbabilidad", se tomaron los resultados arrojados por "entrenamientoOffline" y se observó el resultado que arrojaba la misma.

La prueba que se le realizó a "escribirArchivoOffline", consistió en tomar el resultado arrojado por "evaluarProbabilidad" y observar el archivo de salida que escribía dicha función.

Modo "Online"

Al igual que el modo "Offline", el modo "Online" esta conformado por una función,"entrenarOnline", que se encarga de hacer la llamada a función del resto:"entrenamientoOnline","escribirArchivoOnline" Las pruebas de "entrenamientoOnline" consistieron en darle un modelo de normalidad y un conjunto de estructura de tipo "UriSegmentado" y observar la salida que arrojaba la misma.

Para probar la función "escribirArchivoOnline", se tomó el resultado arrojado por "entrenamientoOnline" y se observó el archivo de salida escrito por la misma.

5.5.2.5. Módulo de evaluación

El modulo de evaluación consta de una función general, llamada "evaluacion" que se encarga de hacer la llamada a función de: "epsiloSumatoria", "calcularIndiceAnormalidad", "evaluarIndiceAnormalidad" y "escribirReporte", que a su vez se encargan de realizar el trabajo del módulo de evaluación. Para probar la función "epsiloSumatoria", se ingresó un conjunto de datos de tipo de tipo "UriSegmentado", un modelo de normalidad previamente leído, un valor para el epsilon, y un conjunto de estado del autómata para observar los resultados que arrojaba la misma. Para la prueba de la función "calcularIndiceAnormalidad", se tomaron los resultados proporcionados por la función "epsiloSumatoria" y se observará el resultado obtenido.

Tanto los resultados obtenidos por "epsiloSumatoria" por "calcularIndiceAnormalidad" fueron verificados calculando de manera manual los valores que estas funciones calculan, haciendo uso de las expresiones correspondientes. Las pruebas de la función "evaluarIndiceAnormalidad" consistieron en darle un índice de anormalidad y un valor del parámetro θ, como los que se muestran en la figura 5.12 y verificar si la misma clasificaba de manera adecuada. Finalmente, para realizar la prueba de "escribirReporte" se le pasó una serie de URIs pertenecientes a la base de datos **db1.pcap**. Una vez realizado esto, se observó el archivo de salida escrito por dicha función.

5.5.3. Pruebas operativas

En esta sección se explicaran las pruebas operativas que se les aplicaron a las tres modalidades del sistema implementado.

5.5.3.1. Modo entrenamiento "Offline"

Para probar el modo "Offline" del sistema, se tomó la base de datos **db3.pcap** explicada en la sección 5.5.1, y se le ingresó al sistema configurado en modo "Offline". Una vez realizado esto, se observó el modelo de normalidad arrojado por

Estado	Palabra	Probabilidad
Bss	192.168.3.48	1.0
Bsp	tr-tema-16.pdf	0.25
Bsp	html	0.25
Bsp	/	0.25
Bsp	tr-problemas1-14.pdf	0.25
numeroPalabraSv	numTotal	0.0
numeroPalabraSs	numTotal	4.0
numeroPalabraSp	numTotal	4.0
numeroPalabraSa	numTotal	0.0

FIGURA 5.23: Modelo de normalidad obtenido en el modo "Offline".

dicha modalidad. En la figura 5.23 se puede apreciar el modelo de normalidad obtenido por la misma.

5.5.3.2. Modo entrenamiento "Online"

El modo "Online" del sistema fue probado ingresando la base de datos **db4.pcap** explicada en la sección 5.5.1 al sistema junto con el modelo mostrado en la figura 5.23. Una vez realizado esto, se observo la salida y se pudo verificar que, en efecto el modelo antiguo fue modificado. En la figura 5.24 se puede apreciar la salida de esta modalidad tras la prueba realizada.

5.5.3.3. Modo evaluación

Para realizar la prueba del modo de evaluación se tomó la base de datos **db1.pcap** para realizar un modelo de normalidad haciendo uso del modo de entrenamiento "Offline". De esta manera, se evaluarán las trazas que se encuentran en la base de datos, **db2.pcap** y **db3.pcap** y se observaran los resultados obtenidos. La intención de esta prueba es comparar los índices de anormalidad obtenidos por la base de datos **db2.pcap** y **db3.pcap**. Como la base de datos **db2.pcap** posee el mismo comportamiento que el del modelo de normalidad por pertenecer a peticiones realizadas al mismo servidor, se espera que los índices de anormalidad sean mas bajos que los obtenido por **db3.pcap**, ya que esta trazas pertenecen

Estado	Palabra	Probabilidad
Bss	192.168.3.48	1.0
Bss	com	0.1666
Bss	ethereal	0.1428
Bss	www	0.2
Bsp	tr-tema-16.pdf	0.25
Bsp	html	0.25
Bsp	/	0.25
Bsp	tr-problemas1-14.pdf	0.25
Bsp	download.html	0.2
Bsp	tr-tema3-16.pdf	0.25
numeroPalabraSv	numTotal	0.0
numeroPalabraSs	numTotal	7.0
numeroPalabraSp	numTotal	5.0
numeroPalabraSa	numTotal	0.0

FIGURA 5.24: Modelo de normalidad obtenido en el modo "Online".

URI	Índice de anormalidad
192.168.3.48/html/	36.0
192.168.3.48/	25.853949
192.168.3.48/	36.0
192.168.3.48/	36.0

FIGURA 5.25: Índices de anormalidad obtenidos de db3.pcap.

a peticiones que no se encuentran registradas en el comportamiento del modelo de normalidad. En la figura 5.25 se muestran los resultados de los índices de anormalidad calculados cuando se evalúa la base de datos **db3.pcap**. Se puede observar que los índices oscilan entre 25 y 36. Por otra parte, en la figura 5.26 se pueden detallar los índices de anormalidad resultantes, de evaluar la base de datos **db2.pcap**.Se puede apreciar que los índices de anormalidad de la figura 5.26 son mas bajos que los de la figura 5.25. Estos resultados son los esperados, ya que **db2.pcap** no presenta incongruencias con el modelo de normalidad construido, mientras que la base de dato **db3.pcap** si. Por lo tanto es conveniente que el sistema lo catalogue como anomalías. Un parámetro θ, óptimo para esta prueba en particular, debería ser un valor que sea menor que 25 y mayor que 18.

URI	Índice de anormalidad
192.168.0.193/es/inicio	5.25
192.168.0.193/es/quienessomos/origen	17.83
192.168.0.193/es/quienessomos/area	17.83
192.168.0.193/es/quienessomos/organigrama	17.83
192.168.0.193/static/images/quienessomos/libro.jpg	15.53
192.168.0.193/static/images/quienessomos/organigrama.jpg	15.53
192.168.0.193/es/patrimonio/nota	13.86
192.168.0.193/es/educacion/escuela	17.83
192.168.0.193/es/educacion/convenios	17.83
192.168.0.193/es/patrimonio/monumentos	13.86

FIGURA 5.26: Índices de anormalidad obtenidos de db2.pcap.

Capítulo 6

CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentarán las conclusiones y los aportes realizados por este proyecto. Así mismo, se mostrará una lista de recomendaciones, que expone un conjunto de ideas de futuras mejoras que se pueden aplicar al proyecto.

6.1. Conclusiones

En el presente proyecto se realizaron todos los objetivos planteados de manera satisfactoria. En primer lugar, se realizó un estudio de SSM ("Stochastic Structural Model") [1] y todos sus conceptos asociados. Una vez realizado este investigación, se diseñó una arquitectura modular del sistema, en la cual se propuso la construcción de tres módulos: uno para capturar, filtrar y segmentar los paquetes de tipo *GET* del protocolo *HTTP*; otro módulo que permite evaluar el índice de anormalidad de los URI y un tercer módulo para estimar los modelos de normalidad. Luego de modelar la arquitectura del sistema, se implementó cada una de las funciones del mismo haciendo uso de la herramientas y del lenguaje de "scripting" de Bro, con la intención de obtener un IDS basado en SSM que puede ser utilizado por servicios web en producción. Por último, se realizaron pruebas tanto funcionales como operativas.

Los aportes realizados por este proyecto son los siguientes:

1. Se ha implementado un sistema que realiza detección de intrusiones en los servidores web.
2. Se ha implementado un sistema con tres funcionalidades operativas: una para realizar detección de intrusiones y dos para realizar modelos de normalidad de los servicios web.
3. Se han construido modelos de normalidad de servicios web que podrían ser utilizados para la detección de intrusiones.

6.2. Recomendaciones

1. Calcular de manera experimental los valores óptimos de θ y los de probabilidad fuera de vocabulario que generen la menor cantidad posible de falsos positivos, haciendo uso de bases de datos mas extensas.
2. Agregarle al IDS un conjunto de firmas de ataques conocidos aplicando la técnica propuesta en [27].
3. Probar el sistema en un entorno de producción.
4. Ampliar el sistema para que funcione en otros protocolos de la red.

Bibliografía

[1] Pedro García-Teodoro.; Jesús E. Díaz-Verdejo; Juan M. Tapiador; Rolando Hernandez-Salazar Automatic Generation of HTTP Intrusion Signatures by Selective Identification of Anomalies Computers & Security, Vol. 55, pp. 159-174, 2015, ISSN: 0167-4048.

[2] PWC: PricewaterhouseCoopers, https://www.pwc.com/gx/en/services/advisory/forensics/economic-crime-survey/cybercrime.html

[3] Walters R. (2017), *Salary Survey*, disponible en: https://www.robertwalters.co.uk/content/dam/salary-survey-2017.pdf?webSyncID=6d3bc948-9837-8e7b-91d2-a51713c993b7&sessionGUID=0b156ce4-a2a8-3d94-8cd0-15640ae068ac

[4] Sundar Rajan, S., & Krishna Cherukuri, V. (2017).*An Overview of Intrusion Detection Systems* . Disponible en: http://www.idt.mdh.se/kurser/ct3340/ht09/ADMINISTRATION/IRCSE09-submissions/ircse09_submission_18.pdf

[5] R,K. & Indra, A.(2010) *Intrusion Detection Tools and Techniques – A Survey. International Journal Of Computer Theory An Enginering*, 901-906. http://dx.doi.org/10.7763/ijcte.2010.v2.260.

[6] Granada y la Universidad. Disponible en: https://www.ugr.es/universidad/organizacion/granada-y-la-universidad

[7] La Escuela de Informática y Telecomunicación. Disponible en: https://etsiit.ugr.es/pages/escuela

[8] Misión y Visión de la Universidad de Granada. Disponible en: http://www.ugr.es/~rhuma/sitioarchivos/noticias/MisionVision.pdf

[9] Departamentos que imparten docencia en la ETSIIT. Disponible en: `https://etsiit.ugr.es/pages/escuela/departamentos`

[10] H. Debar (2000), *An Introduction to Intrusion-Detection Systems*, IBM Research, Zurich Research Laboratory,Ruschlikon, Switzerland.

[11] RFC 7230

[12] RFC 3986

[13] Linz P. (2012) *An Introduction to FORMAL LANGUAGE an AUTOMATE*, quinta edición, pp 36.

[14] Mishra K.L.P. , Chandrasekaran N. (2008), *THEORY OF COMPUTER SCINCE Automata, Language and Computation*,tercera edición, pp 71-72.

[15] V. Aho A., S. Lam M., Sethi R., D. Ullman J. (2007),*Compilers, Principles, Techniques, & Tools*, segunda edición,pp 147-150.

[16] Ching W., Ng M. K. (2006), *Markov Chains: Models, Algorithms and Applications*, pp 1-4.

[17] Haggstrom (2002) *Finite Markov Chains and Algorithmic Applications*, Lon- don Mathematical Society, Student Texts 52, Cambridge University Press,Cambridge, U.K.

[18] Rolando Salazar Hernández, Sistema de detección de intrusos mediante modelado de URI. Tesis doctoral. Universidad de Granada. Director: Jesús E. Díaz Verdejo, 02/02/2016

[19] *About - GitHub*. Disponible en: `https://github.com/about`

[20] *GitHub Glossary*. Disponible en: `https://help.github.com/articles/github-glossary/`

[21] *Wireshark - about*. Disponible en : `https://www.wireshark.org/about.html`

[22] *Wireshark - docs*. Disponible en: `https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html`

[23] *Bro Introduction*, disponible en : `https://www.bro.org/sphinx/intro/index.html`

[24] Kurose, J. F., & Ross, K. W. (2013). *Computer networking: a top-down aproach*. Pearson

[25] Cormen H. C., Leiserson E. C., Rivest L. R., Stein C. (2009).Introduction to algorithms, tercera edición, pp 153-155.

[26] Capturas de ejemplo de "Wireshark". Disponible en: `https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=http.cap`

[27] García-Teodoro P., Díaz-Verdejo J.E., Tapiador J.E., Salazar-Hernández R. (2015) *Automatic generation of HTTP intrusion of anomalies*