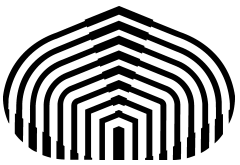


Universidad Simón Bolívar
Decanato de Estudios Profesionales
Coordinación de Ingeniería de la Computación

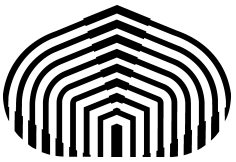
@títuloProyecto

Por:
@autor1
@autor2

Realizado con la asesoría de:
@tutor

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Computación

Sartenejas, @mes de @año



UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS PROFESIONALES

COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PROYECTO DE GRADO

@tituloProyecto

Presentado por:

@autor1

@autor2

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

_____

@tutor

_____

@jurado1

_____

@jurado2

Sartenejas, @día de @mes de @año

Resumen

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Palabras clave: @palabra1, @palabra2, @palabra3.

Agradecimientos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Índice general

Índice de figuras

Índice de Tablas

Acrónimos y Símbolos

SIGLAS	**S**iglas **I**sla **G**rafo **L**aos **A**ve **S**erpiente
ACM	**A**ssociation for **C**omputing **M**achinery

$\Longleftrightarrow$	doble implicación, si y sólo si
$\Rightarrow$	implicación lógica
$[u := v]$	sustitución textual de u por v

Dedicatoria

A @personasImportantes, por @razonesDedicatoria.

Introducción

Hoy en dia es muy difícil imaginar el mundo sin el Internet. Esta gran red de redes se ha convertido con el pasar de los años en la espina dorsal del mundo y se estima que mas y mas dispositivos se iran uniendo a esta en un futuro.

Un gran porcentaje del uso que se le da al Internet proviene de las aplicaciones web. Las aplicaciones web son programas cliente-servidor en donde el cliente se corre en un web browser. Estas pueden ser desde correos electronicos, hasta paginas web con algun servicio en especifico. Para compartir toda la informacion brindada por estas aplicaciones a traves del Internet, se hace uso del protocolo HTTP (Hypertext Transfer Protocol). Es aqui donde entran los servidores HTTP, estos servidores son los encargados de suplir las peticiones de sus clientes y enviar o recibir la informacion 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 aplicacion que contiene las claves, los nombres de usuarios y numero de tarjetas de credito.

Una gran cantidad de informacion esta depositada en servidores de este tipo, es por esto, que es de total importancia velar por su seguridad.

Este tema en cierta medida será el tema central en el presente trabajo. En el mismo, se hablara sobre la implementación de un protocolo desarrollado por J.E. Díaz-Verdejo, P. García-Teodoro, P. Muñoz, G. Maciá-Fernández, F. De Toro de la Universidad de Granada llamado SSM (Segmented Stochastic Modelling) en un lenguaje se scripting de un analizador de redes llamado Bro.

SSM (Segmented Stochastic Modelling) es un sistema que "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" J.E. Díaz-Verdejo, P. García-Teodoro, P. Muñoz, G. Maciá-Fernández, F. De Toro. 2007. Una aproximación basada en Snort para el desarrollo eimplantación de IDS híbridos.

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.

En el siguiente trabajo se explicarán los detalles de la implementación del sistema, sus bases teóricas y las pruebas realizadas sobre el mismo. La estructura de este será la siguiente:

En el capitulo uno se hablará sobre la motivación de realizar dicho proyecto, los objetivos tanto generales como especificos, la planificación que se utilizo para realizar el proyecto y el presupuesto del mismo.

Por otra parte,en el segundo capítulo se tocarán temas relacionados con el estado del arte del proyecto. En primero lugar se explicara la definición de IDS y los tipos de IDS que existen, se explicara 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 hablará sobre la herramiento utilizada y el lenguaje de scripting se utilizó para implementar el sistema.

En el tercer capitulo se hablara sobre el diseño del sistema. Aquí se describirán los modulos en los que esta dividido el IDS y como es el funcionamiento de cada uno. Los detalles de implementacion de estos se explicaran en el capitulo cuarto.

El quinto capitulo tocará el tema de la evaluación y las pruebas. En este capitulo se hablara sobre la base de datos utilizada para realizar las pruebas y los resultados arrojados por las mismas. Además se describiran un poco las pruebas tanto operativas como funcionales realizadas sobre el sistema.

Capítulo 1

Introducción

1.1. Objetivos generales

1. Implementar un detector de ataques híbrido que haga uso del sistema SSM mediante el uso de la herramienta Bro.

1.2. Objetivos especificos

1. Implementación de una representación de un autómata de estados finito que sirva para evaluar tanto la sintaxis como la probabilidad de generación de una petición del protocolo HTTP.

2. Desarrollo de un modulo que mida la probabilidad de generación de los segmentos del URI.

3. Desarrollo de un modulo de entrenamiento con el cual se permita obtener un modelo de normalidad a partir de trafico libre de ataques.

4. Obtener los valores optimos del los parametros de configuracion del sistema para asi garantizar la menor cantidad de falsos positivos posibles.

1.3. Planificación

El desarrollo de este proyecto se basó en cuatro grandes fases:

1. Conocer el estado del tema.
2. Implementación del autómata de reconocimiento de URI's
3. Realizar los entrenamientos.
4. Documentación y desarrollo de libro de pasantía.

En la fase de conocer el estado del arte del tema del proyecto tuvo una duración de cuatro semanas y durante este tiempo se estudio la documentación de la herramienta BRO y su lenguaje de scripting. En esta subfase, se aprendió a utilizar tanto la herramienta BRO como a programar haciendo uso de su lenguaje de scripting. Además, se hizo un estudio sobre los IDS, los tipos de IDS y el sistema SSM. Este fue un paso muy importante dentro de la documentación ya que el hecho de entender que era un IDS y como funciona un sistema SSM era una base fundamental para la realización del proyecto. Así mismo, se hizo una lectura del RFC 3986 (URI) y se realizó una pequeña investigación sobre el protocolo HTTP.

Luego de la fase de conocer el estado del arte se procedió a la segunda fase: la implementación del autómata de reconocimiento de URIs. En esta fase se desarrolló el filtro de peticiones GET del protocolo HTTP en el lenguaje de scripting de Bro, y luego se implementó un modulo para segmentar los URIs que venían con las peticiones de tipo GET. Finalmente, se desarrolló el modulo de evaluación cuya función es la de evaluar la probabilidad de generación de los segmentos del URI dado un modelo de normalidad. Esta fase duró seis semanas y para ambos modulos implementados se realizaron pruebas funcionales.

Después de culminar la fase anterior se procedió a abordar la fase de realizar los entrenamientos. Esta fase consistió en implementar el modulo de entrenamiento del sistema para poder realizar modelos de normalidad del mismo, obtener paquetes de tipo http haciendo uso de un "sniffer"para realizar un pequeño modelo de

normalidad y verificar la correctitud del modulo implementado. Luego se procedió a realizar pruebas con las bases de datos RDB que es una base de datos de ataques, PVHR21 Y PVHR22 que son bases de datos con trazas normales, con la intención de probar el funcionamiento global del sistema

Una vez probada toda la correctitud del sistema se procedió a desarrollo de libro del proyecto de grado. Esta fase consistió en estructurar y redactar el libro del proyecto.

1.4. Presupuesto

Capítulo 2

Marco teórico

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

En líneas generales, el trabajo, tiene como propósito explicar la implementación de un *IDS* híbrido que haga uso del sistema *SSM* a traves del lenguaje de scripting de la herramienta *BRO*.

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

2.1. Detector de intrusiones (IDS)

Un detector de intrusiones o IDS por sus siglas en inglés (intrusion-detection system) a manera general es un sistema que se encarga de procesar la información entrante de un sistema a proteger.

La intención de tomar la información entrante es la de realizar un diagnostico de seguridad para así descubrir los posibles ataques, brechas de seguridad o vulnerabilidades que pueden existir en el sistema que se quiere proteger.

Existen diversos tipos de IDS. Pero los mas comunes son los IDS basados en firmas, los IDS basados en comportamiento y los IDS híbridos que son una mezcla entre un IDS basado en firma y un IDS basado en comportamiento.

Los IDS basados en firmas, utilizan una base de datos que contiene una lista de posibles ataques que pueden ser perpetuados y las vulnerabilidades del sistema. Entonces, el IDS cuando esta filtrando la información que va a entrar al sistema compara esa o parte de la informacón con la base de datos de ataques, si dentro de la información entrante existe una firma que se encuentra en la base de datos entonces se activará una alarma para indicar que un ataque esta siendo perpetuado. De otro modo, la información entrante será catalogada como aceptable.

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, están los IDS basados en comportamiento que detectan los ataque de manera diferente. En este tipo de IDS los ataques son detectados a partir de la observación del comportamientos, bien sea del sistema o de los usuarios.

El modo de funcionamiento de un IDS basado en comportamiento se basa en, recolectar la información entrante del sistema a proteger 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 muy grande entre la información entrante y el modelo de normalidad, entonces se generará una alarma.

No obstante, uno de los problemas que posee este tipo de detector de instrusiones es la alta tasa de falsos positivos que puede llegar a tener. Esto se debe a que es casi imposible que un modelo posea todos los comportamientos normales de un sistema en su totalidad, lo cual provocaría que informacón que está libre de ataques sea catalogada como una amenaza. También esta 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 del y reentrenamiento modelo de normalidad el mismo quedará obsoleto y la información entrante será mal catalogada por el IDS. El problema de realizar entrenamientos constantes para actualizar un modelo de normalidad es que, a pesar de que el modelo contendrá una información acertada acerca del comoportamiento del sistema se crearán brechas de tiempo en donde el sistema será muy vulnerable a ataques ya que en lugar de estar funcionando el modulo para detectar ataques estará funcionando el modulo para entrenar el modelo de normalidad. Por lo tanto, si un ataque es perpetuado durante ese periodo de tiempo el mismo quedará grabado en el modelo de normalidad como un comportamiento normal del sistema.

Así mismo también existen los IDS híbridos cuyo funcionamiento mezcla el funcionamiendo de los detectores de instrusiones basados en firma y los basados en comportamiento. En pocas palabras, los IDS híbirdos suelen contar con una base de datos de ataques y también con un modelo de normalidad.

El IDS implementado es de típo híbrido ya que por una parte, se revisan las firmas (en este caso los URI) de la información entrante, y si estas no muestran ninguna incongruencia, dicha información pasará a ser contrastada con el modelo de normalidad. Sıla misma es muy diferente a dicho modelo, entonces se emitirá una alarma.

2.2. Protocolo HTTP

Definición:

Hypertext Transfer Protocol (HTTP) según el RFC 7230 es un protocolo cliente/servidor que funciona a nivel de capa de aplicación y cuya función es transferir hipertexto a traves de la red.

Este protocolo es considerado una de las bases fundamentales dentro de la comunicación de datos en el Internet.

El cliente de un protocolo HTTP es un programa que se encarga de crear una conexión con el servidor y enviar las diferentes peticiones de recursos al mismo. Por su parte, el servidor HTTP se encargará de aceptar dicha conexón y responder a las peticiones hechas por el cliente.

Los recursos de este protocolo son identificados por un Uniform Resource Identifier (URI).

Por otra parte, este protocolo posee diferentes metodos para solicitar las peticiones de recursos. Estos son, el método GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE.

Los métodos GET, HEAD y POST son los mas utilizados en las comunicaciones del protocolo HTTP. El método GET, consiste en solicitar de un cierto recurso al programa servidor HTTP. Por su parte,el método POST se utiliza para informar al destinatario que procese la información que viene incluida en la solicitud realizada y el método HEAD, funciona de manera identica al método GET con la única diferencia de que la solicitud solo será respondida con la cabecera del recurso excluyendo el cuerpo del mismo.

2.3. URI

Según el RFC 3986 un URI (Uniform Resource Identifier) es una serie de caracteres que identifican un recurso en la red. Este tiene una sintaxis especifica que esta conformada por diferentes segmentos de caracteres para el esquema, la autoridad, la ruta, el query y el fragment.

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

- Esquema:

El esquema identifica el protocolo que va a ser utilizado. En el caso del presente trabajo, los URIs utilizaran unicamente el protocolo http o https.

- Authority:

  Este componente del URI posee información del usuario que pueden ser nombres de usuarios y contraseñas (campo opcional), el host y el puerto correspondiente (campo opcional).

  El host de un URI viene representado bien sea por un IPv4 o un nombre de dominio, seguido de un número de puerto (opcional) que identifican la máquina en donde están los recursos a solicitar.

- Path: La ruta de un URI son un conjunto de segmentos organizados de manera jerarquica y separados por slashes 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 simbólo de inicio es el signo de interrogación (?). Por lo general, el query está conformado por una dupla .ªtributo=valor"que junto con la ruta ayudan a identificar el recurso que se desea solicitar. No obstante, a diferencia de la ruta, la información aquí contenida debe ser procesada por el servidor al que se le esta solicitando el recurso.

  Por otra parte, el atributo, representa el nombre de una variable y el valor vendria siendo el valor que contendrá dicha variable.

- 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 simbolo 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 caracter vacío, se asumirá que la misma es /". El resto de los segmentos son opcionales.

En la figura 2.1, se muestra, en un ejemplo los componentes que forman un URI.

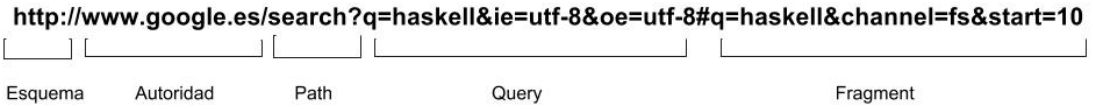


FIGURA 2.1: URI.

2.4. Automátas de estados finitos

Los automatas son modelos muy importante dentro de las ciencias de la computación. Su presencia está en diversos ambitos de esta ciencia. Desde programas para hacer la verificación de protocolos de comunicación, analizadores sintácticos de compiladores,programas que realizan la verificación del comportamiento de circuitos digitales o el escaneo de largos bloques de texto.

Antes que nada es preciso recalcar que existen dos tipos de automatas de estados finitos: los automatas deterministas y los no deterministas.

Los automatas deterministas son aquellos que para cada input solo existe un estado del automata al cual se puede transitar.

Formalmente, un automata determinista, A, se puede representar de la siguiente manera:

A = (Q,Σ, δ, q_0, F)

Donde:

- Q : Es un conjunto de estados.
- Σ : Es un conjunto de simbolos, tambien llamado alfabeto.
- δ : Es una función de transición que toma como argumento un estado y un simbolo y retorna un estado, es decir, δ : Q x $\Sigma \rightarrow Q$. Si el automata es representado como un grafo, entonces esta función de trasnsición vendria representada por el arcco que une dos nodos junto al simbolo del mismo.
- q_0: Es el estado inicial del automata.

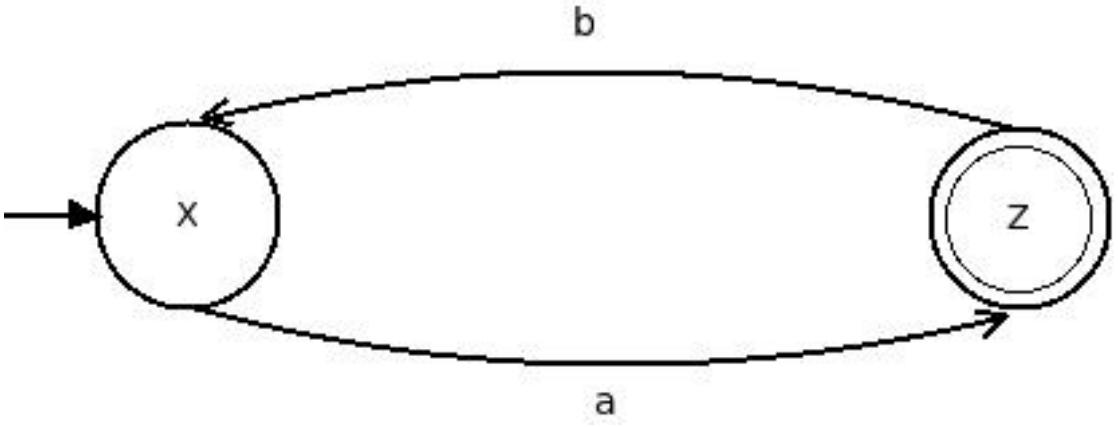


FIGURA 2.2: Automata determinista.

- F : Es un conjunto de estados finales.

A continuación se presentará un ejemplo de un automata.

A = ({X,Z},{a,b},{$\delta(X, a) = Z, \delta(Y, b) = X$},{X},{Z})

El mismo será representado a traves de un grafo, en donde los nodos representarán los estados del automata, y los arcos junto a sus simbolos representaran las funciones de transición en la figura 2.2.

Por otra parte, los automatas no deterministas, a diferencia de los automatas deterministas pueden tener un conjunto de estado en el automata al cual se puede transitar para cada input, es decir, la función de transición de un automata no determinista en lugar de devolver solo un estado dado un simbolo y un estado puede devolver un conjunto de estados.

La representación de los mismos es analoga a la de los automatas deterministas con la diferencia que δ estaría definida como una función de transición de la siguiente forma δ : Q x $\Sigma \rightarrow \{Q\}$.

2.5. Modelo de Markov

Un modelo de Markov los eventos futuros dependen de los eventos anteriores, es decir, son procesos con memoria.

Un modelo de Markov discreto, λ, se define de la siguiente forma:

λ = (Q,Θ,A,B,Π)

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. A = $[a_{ij}], 1 \leq i \leq Ny1 \leq j \leq N$.
  a_{ij} = P($q_t = S_j$ — $q_{t-1} = S_i$)
- B es una matriz NxM de probabilidades de generación u observaciones de los símbolos entre estados.
  B = $[b_{ik}], 1 \leq i \leq Ny1 \leq k \leq M$.
  b_{ik} = P($O_t = v_k$ — $q_t = S_i$)
- Π es el vector de probabilidades del estado inicial. $\Pi = [\Pi_i], 1 \leq i \leq N$
  π_i = P($q_1 = S_i$)

2.6. SSM

SSM o Structural Stochastic Modeling en inglés es una tecnica desarrollado por Estevez y 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" J.E. Díaz-Verdejo, P. García-Teodoro, P. Muñoz, G. Maciá-Fernández, F. De Toro. 2007. Una aproximación basada en Snort para el desarrollo e implantación de IDS híbridos.

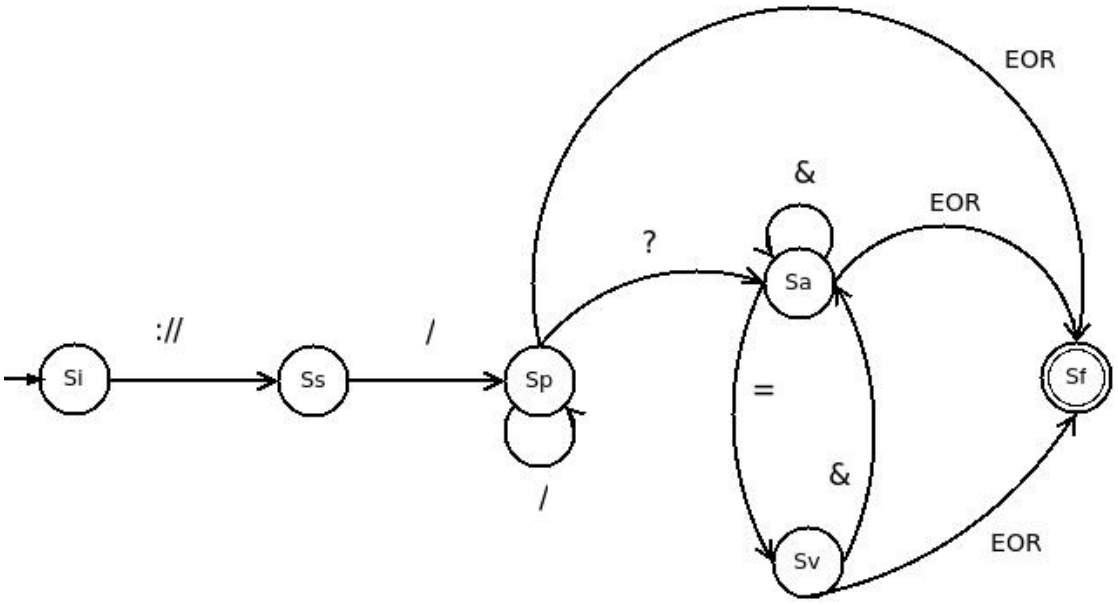


FIGURA 2.3: Automata del modelo SSM

Es importante recalcar que esta técnica además se basa en la teoría de los modelos de Markov ya que se define un automata de estados finitos y a partir de este se permite evaluar y saber cual es 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, el tipo de peticiones se puede extender para que funcione con métodos de tipo POST y HEAD.

Este hecho hace que el automata de SSM deba tomar una cierta topología para que de esta forma se puedan reconocer los URIs de cada petición. Dicha topología se infiere con ayuda de la sección 2.3, en donde se explica tanto la sintáxis de los URIs como el modo de segmentar los mismos. En la figura 2.3 se muestra la topología del automata inferida.

Las transiciones del automata vienen dada por las especificaciones de las sintaxis de los URIs que están descritas en el RFC 3986. Además, se puede observar en la imágen que el número de estados es casi el mismo que el número de segmentos de un URI que se presenta en la sección 2.3 . En el automata, se 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 terminaŕa si el URI no es sintácticamente correcto.

Si en una petición del protocolo HTTP viene anexado un URI que no puede ser reconocido por el automata descrito con anterioridad implicará que el URI de dicha petició no está bien construido sintacticamente.

Además de esto, existirá un vocabulario diferente para los estados S_S, S_P, S_A y S_V en donde se encuentren las posibles palabras que puedan aparecer en los mismos.Estos vocabularios son construidos a partir de tomar múltiples peticiones libres de ataques realizadas al servidor.

Entonces, en pocas palabras lo que hace la técnica de SSM para detectar intrusiones es verificar la correctitud de la sintaxis del URI de las peticiones enviadas al servidor a traves del automata 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 automata para así verificar si el URI que viene anexado al la petición es una cadena de caracteres que probabilisticamente corresponde a un petición normal o no.

Por lo tanto, el modelo teorico utilizado por SSM es el siguiente:

- 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.

2.7. Bro

Bro es un software open source capáz de analizar a detalle las actividades que ocurren en la red.

Por otra parte, este software incluye un lenguaje de scripting orientado a eventos cuya función es extender y personalizar las funcionalidades primarias que otorga Bro.

Este lenguaje de scripting cuenta con una cola de eventos de tipo FIFO y un manejador de eventos que se encarga de servir a los mismos.

Cuenta, ademá con un tipo de registro especial que almacena la información del estado de las conexiones durante su tiempo de vida.

Por otra parte, dispone de los tipos de datos comunes que encontramos en la mayoría de los lenguajes de programación como el ïnt", que representa los números enteros de 64 bits, "double"que representa los números flotantes de 64 bits, "bool"que representa los booleanos y çount"que representa los números enteros sin signo de 64 bits. Además de estos tipos de datos el lenguaje de scripting de Bro posee otro tipos de datos mas especificos para poder trabajar el análisis de redes de manera mas cómoda como son el .ªddr"que es un tipo de datos destinado a las direcciones ip, "port"para los puertos de la capa de transporte, "subnet"para las mascaras de subred, "time"para almacenar datos que representen tiempo, ïnterval"para representar intervalos de tiempo y "pattern"para las expresiones regulares.

Además, tine estructura de datos como: conjuntos, tablas, vectores y registros.

Así mismo, el lenguaje de scripting de Bro posee un framework que permite manejar de manera cómoda y sencilla los logs del sistema y otro para leer de manera eficiente la información dentro de los mismos. Con el primero, se puede crear archivos, escribir datos de forma organizada y filtrar informacón de los logs. El segundo framework lee la información que se encuentra dentro de los logs para luego ser almacenada en tablas o activar cierto evento anteriormente programado.

Capítulo 3

Arquitectura del IDS

En esta sección se explicará la arquitectura y el diseño del IDS implementado. La arquitectura del detector de instrusiones que se muestra en el presente trabajo se basa en una arquitectura modular.

Este consta de tres modulos. Un modulo para realizar la segmentación de los URIs, otro modulo para realizar la evaluación y un último modulo para realizar el entrenamiento y de esta forma crear un modelo de normalidad.

A grandes rasgos, el modulo de segmentación se encargará de tomar el URI que proviene de la solicitud de tipo GET que se le hace al servidor HTTP, lo normalizará, lo segmentará siguiendo las especificaciones del RFC (insertar nombre del RFC) y lo almacenará en una estructura de datos para que de esta forma pueda ser utilizado en los otros dos modulos.

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 modulo de segmentación para al final decidir si el URI de la solicitud enviada al servidor de tipo HTTP 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.

3.1. Modulo de segmentación

Como se mencionó con anterioridad, el modulo de segmentación es el que se encarga de segmentar los URIs recibidos en las solicitudes de tipo GET o HEAD por el servidor de tipo HTTP. Pero antes de segmentar el URI, es necesario normalizarlo. Es por ello, que a continuación se describirá como es el proceso tanto de normalización como de segmentación de un URI.

3.1.1. Normalización de los URIs

La normalización de los URIs es un paso importante dentro del modulo de segmentación ya que mediante este paso es posible estandarizar todas las posibles formas en las que se puede escribir un URI en una para de este modo facilitar tanto la evaluación como el entrenamiento en el sistema.

La normalización en pocas palabras, consiste en tomar el URL en el formato en el que viene y codificarlo al formato de tipo UTF-8. Este paso evitará inconvenientes al momento de guardar palabras o frases que sean iguales de manera doble en el diccionario solo porque las mismas estan codificadas siguiendo diferentes formatos.

Un ejemplo sencillo de lo que se haría en la normalización del sistema se mostrará a continuación:

Supongamos que al servidor de tipo HTTM le llega una petición de tipo GET con el siguiente URL: https://192.168.0.23?q=security+network

Pero luego de un tiempo este mismo servidor recibe otra solicitud tipo GET con el siguiente URL:

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

Es claro que ambos URL poseen la misma información, y que a grandes rasgos se podría decir que son iguales. No obstante, el espacio en blanco esta codificado de manera diferente en cada uno de los dos. Esto significa que si no se normalizan los URIs la palabra "security+networkz "security %20network"serían

dos palabras totalmente disferentes dentro del vocabulario del sistema cuando no debería ser así. Entonces, en este caso, se normalizaría el primer URI para que el mismo este codificado en formato UTF-8 para que de esta forma se parezca al segundo URI (https://192.168.0.23?q=security %20network). Así, la frase "security+network"pasaría a ser "security %20network".

3.1.2. Segmentación de los URIs

Para la implementación del detector de instrusiones que se explica en el presente trabajo es de suma importancia realizar la segmentación y almacenamiento de los URIs que vienen adjuntos en las solicitudes que se le realizan al servidor HTTP.

La idea del segmentador de URI es considerar una serie de delimitadores, que delimitan areas especificas dentro del URI y almacenar cada uno de los tokens que son divididos por estos de forma organizada en una estructura de datos.

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 : Esta primera parte corresponderá al segmento de protocolo.
- 159.90.9.166 : Esta segunda parte será el segmento del host.
- consulta : Esta tercera parte vendría siendo parte del segmento de recursos.
- manifestacion :Esta parte correspondería al atributo
- Pinturas %20Rupestres : Esta parte corresponde al valor.

Cada uno de los segmentos serán almacenados en una estructura de datos para ser utilizados posteriormente en el sistema.

3.2. Modulo de evaluación

Este modulo, como se mencionó con anterioridad es el encargado de evaluar la probabilidad de generación de cada uno de los segmentos del URI dado un modelo de normalidad. Una vez calculadas estas probabilidades, el modulo se encargará de calcular un indice de anormalidad del URI mediante el uso de las formulas descritas en la sección (inserte número de la sección) para luego comparar dicho valor obtenido con un parámetro θ. Si el indice de anormalidad es mayor o igual que el parametro θ, entonces se dirá que el URI es anomalo. Por el contrario, si el indice de anormalidad es menor que dicho parámetro, entonces, se podría decir que el URI no posee anomalías.

3.2.1. Expresiones para calcular indice de anormalidad

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:

$$P(O|\lambda, Q) = \pi_{q1} b_{q1o1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}} b_{q_{t+1} o_{t+1}} \tag{3.1}$$

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

$$log(P(O|\lambda, Q)) = log(\pi_{q1} b_{q1o1} \prod_{t=1}^{T-1} a_{q_t q_{t+1}} b_{q_{t+1} o_{t+1}}) \tag{3.2}$$

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 (inserte número de la fórmula anterior) se podría resumir en la siguiente:

$$P(O|\lambda, Q) = \sum_{t=1}^{T} b_{q_t o_t} \tag{3.3}$$

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:

$$N_s = -log(P(U|\lambda)) \tag{3.4}$$

La fórmula nombrada con anterioridad 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

$$N_s = \begin{cases} -log(P(U|\lambda)) & \text{si } q_t = S_f \\ \infty & \text{si } q_t \neq S_f \end{cases} \tag{3.5}$$

Como se puede observar en la fórmula presentada con anterioridad, 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:

$$Clase(U) = \begin{cases} Normal & \text{si } N_s(U) \leq \theta \\ Anomalo & \text{si } N_s(U) > \theta \end{cases} \tag{3.6}$$

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 (insertar número de la fórmula) posee algunos inconvenientes ya que no se estipula el caso en el que existan problemas de entrenamiento insuficiente que será presentado en la sección (inserte número de 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:

$$\varepsilon_0 = E[B] = \frac{1}{M} \sum_{i=1}^{M} b_i \tag{3.7}$$

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(Si): probabilidad de out of

vocabulary) . 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(Si), es cuando la probabilidad de una palabra que está en el vocabulario es inferior al poov(Si), es decir:

$$p_{qtot} = \begin{cases} b_{qtot} & \text{si } o_t \in O \text{ y } b_{qtot} > p_{oov}(q_t) \\ p_{oov} & \text{en otro caso} \end{cases} \tag{3.8}$$

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

$$N_s = \begin{cases} -Tlog(\varepsilon_0) \sum_{t=1}^{T} log(p_{qtot}) & \text{si } q_t = S_f \\ \infty & \text{si } q_t \neq S_f \end{cases} \tag{3.9}$$

3.2.2. Elementos adicionales del modelo

Entrenamiento insuficiente

3.3. Modulo de entrenamiento

El módulo de entrenamiento será el encargado de crear el modelo de normalidad del sistema, es decir, este módulo dará el conjunto de vocabulario junto con la probabilidad de observación de cada una de las observaciones. 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.

3.3.1. Expresiones usadas en el modulo de entrenamiento

Como se dijo con anterioridad, este módulo será el encargado de armar el vocabulario del módulo de normalidad del sistema. A grandes, el módulo de entrenamiento tomará un conjunto de solicitudes libres de ataques, las segmentará, para luego calcular la probabilidad de aparición de las palabras en el conjunto de observaciones totales de cada uno de los estados. En términos más formales, la tarea que realiza este módulo se puede modelar de la siguiente forma:

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 (omega) de L pares de secuencias de estados

$$\omega = (O^1, Q^1), (O^2, Q^2), ..., (O^L, Q^L) \tag{3.10}$$

Tal que

$$\begin{aligned} O^i &= o_1^i, o_1^i, ..., o_{Ti}^i \\ Q^i &= q_1^i, q_2^i, ..., q_{Ti}^i \end{aligned} \tag{3.11}$$

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

La expresión para el vocabulario sería la siguiente:

$$\theta_{Sk} = \bigcup_{i=1}^{L} o_j^i | q_j^i = S_k \tag{3.12}$$

Para calcular la probabilidades de observación se utilizaría la siguiente expresión:

$$b_{ij} = \frac{\sum_{s=1}^{L} \sum_{t=1}^{Ts} \delta(o_t^s = v_{i,j}, q_t^s = s_i)}{\sum_{s=1}^{L} \sum_{t=1}^{Ts} \delta(q_t^s = s_i)} \tag{3.13}$$

Capítulo 4

Detalles de implementación

En el capítulo anterior se pudo apreciar la arquitectura del IDS que se desea implementar haciendo uso del lenguaje de scripting de Bro. Allí se pudo observar que el mismo está constituido básicamente por tres módulos fundamentale: un módulo de segmentación que se encarga de normalizar y segmentar el URI. Un módulo de evaluación que se encargará de evaluar la probabilidad de generación de un URI dado un modelo de normalidad para así clasificar el mismo como normal o anormal y un módulo de entrenamiento cuyo trabajo será crear modelos de normalidad dados un conjunto significante de URI sin alguna anormalidad.

A continuación se explicará las estructuras de datos utilizadas y como se realizó la implementación de cada uno de los módulos mencionados anteriormente haciendo uso de las herramientas presentadas por el lenguaje de scripting de Bro.

4.1. Implementación del modulo de segmentación

A continuación 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. En el capítulo anterior se pudo apreciar la arquitectura del módulo de segmentación. Además, se pudo observar que el mismo consta de dos funcionalidades fundamentales: la normalización de los URIs y la segmentación. A grandes rasgos, la normalización se encargará de tomar el URI de las peticiones HTTP entrantes y codificarlo a formato UTF-8, el output arrojado por esta función será tomado 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 (inserte número de la sección), es decir, el URI se dividirá en las diferentes partes estipuladas en el RFC (insertar número del RFC): el host, el path, los argumentos, los valores y el fragment. Estos segmentos obtenidos luego ser almacenados en una estructura de datos global para que de esta forma, la información obtenida pueda ser utilizada por el resto de los módulos.

La implementación de la función de normalización que se explicará a continuación está basada en el uso de una hashtable con todos los elementos de una encoding table de tipo UTF-8. Por otra parte, la función de segmentación que teóricamente esta basada en un autómata que reconoce el lenguaje de los URI se implementará tomando la inspiración de una gramática que genere el mismo lenguaje que reconoce el autómata (el lenguaje de los URIs). Esto se puede realizar ya que las gramáticas poseen el mismo nivel de expresividad que un autómata, como se mostrará en la secciones próximas. Además, las herramientas presentadas por el lenguaje de scripting Bro facilitan la implementación de un segmentador inspirado en una gramática.

4.1.1. Estructura de datos

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

Función de normalización

La estructura de datos utilizada en la función de normalización es un hashtable 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, ya que lo que se quiere

es 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, como lo explica Cormen en su libro: Ïntroduction to Algoriths":

"Una hash table es una estructura de datos efectiva para implementar diccionarios. A pesar de que la búsqueda de un elemento en una hash table puede tomar el mismo tiempo de búsqueda que una lista enlazada - O(n)/tiempo en el peor de los casos - en la práctica, el mapeo funciona extremadamente bien. Bajos suposiciones razonable, el tiempo de búsqueda promedio de una hash table es de O(1)".

Para implementar este tipo de estructura se utilizó el tipo de dato "table" otorgado por el lenguaje de scripting de Bro. Un ejemplo concreto de como luce la estructura de dato implementada en Bro se mostrará a continuación:

```
global encoding : table[string] of string = {    ["%21"] =    "!"     ,
                                                 ["%22"] =    """    ,
                                                 ["%23"] =    "#"    ,
                                                 ["%24"] =    "$"    }
```

La funcionalidad de esta tabla de codificación será explicada en la sección (insertar sección de implementación de normalización)

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 (insertar número del RFC), y el número de estados que fueron visitados en el autómata presentado en el modelo teórico (insertar número de imagen) para reconocer al mismo .

En Bro existen las palabras claves “record” y “type” y se utilizan de forma similar en las que se emplean las palabras claves “typedef” y “struct” en C. Con estas palabras, Bro permite combinar nuevos tipos de datos y crear tipos de datos compuestos para adaptarse a las necesidades de la situación. A continuación, se mostrará un ejemplo de un tipo de dato compuesto escrito en Bro.

```
type Service: record {
    name: string;
    ports: set[port];
    rfc: count;
};
```

Cuando se combina la palabra clave “type”, el registro puede generar un tipo de dato compuesto.

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. Es necesario crear este tipo de dato ya que se requiere una estructura que almacene los diferentes tipos de datos de los que se hablaron anteriormente.

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 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 uso de esta estructura y el de cada uno de sus campos se explicara a detalle en las secciones siguientes.

4.1.2. Implementación del automata de reconocimiento de URIs

Explicar lo que se debe implementar. Explicar que los automatas son tan potentes y tienen el mismo nivel de expresividad que una grámatica. Explicar que la implementacion se baso en modelar una gramatica con Bro a partir de funciones. EXplicar mas o menos como es el funcionamiento. Explicar como se guardan los parametros.

4.1.3. Implementación del la normalización de los URIs

Explicar que se utilizó la encoding table

4.2. Implementación del modulo de evaluación

4.2.1. Estructura de datos

4.2.2. Lectura del modelo de normalidad en Bro

4.2.3. Evaluación de las probabilidades de los URI

4.3. Implementación del modulo de entrenamiento

4.3.1. Estructura de datos

4.3.2. Entrenamiento Online

4.3.3. Entrenamiento Offline

4.3.4. Escritura del modelo de normalidad en Bro

Capítulo 5

Evaluación y pruebas

5.1. Base de datos

5.1.1. Base de datos normales

5.1.2. Base de datos anomalas

5.2. Experimentación

5.2.1. Pruebas funcionales

5.2.2. Pruebas operativas

Capítulo 6

Conclusiones y Recomendaciones

Apéndice A

@nombreApendice

A.1. @sección

A.1.1. @subsección

"Saludo".

Apéndice B

@nombreApendice