Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Proyecto Tecnológico
Trimestre 2014 - Otoño

Implementación de un NIDS en un sistema embebido para el análisis de tráfico de una red

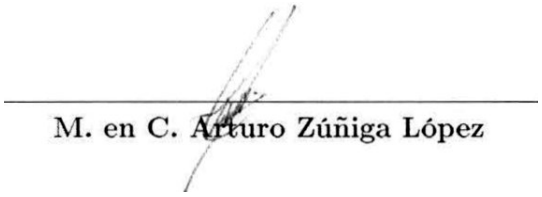
Alumno:
Gibrann Montero Quintero
209300962

Asesor del Proyecto:
M. en C. Arturo Zúñiga López

México, D.F. Diciembre de 2014

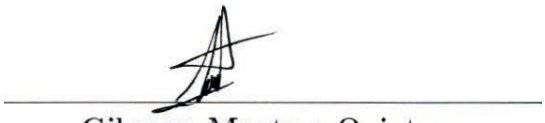
Declaratorias

Yo, **Arturo Zúñiga López**, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



M. en C. Arturo Zúñiga López

Yo, **Gibrann Montero Quintero**, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Gibrann Montero Quintero

Resumen

Un Sistema de Detección de Intrusos en Red **NIDS**, es un sistema que provee de seguridad extra en una red, ayudándose de los equipos de comunicación que son parte de ésta. En la actualidad el número y tipo de ataques que tratan de vulnerar la seguridad de una red, ha aumentado considerablemente, por ello es necesario contar con equipos de seguridad especializados como un NIDS.

La detección de ataques en su gran mayoría, se realiza mediante el análisis de su comportamiento y la comparación de éste con una base de firmas, éste procedimiento es justamente el que se utilizó en este proyecto implementando el Algoritmo de Reconocimiento de Patrones **Naive Bayes**. Dicha implementación del algoritmo se realizó en el lenguaje de programación **Python**.

La finalidad de implementar un NIDS en un sistema embebido es agilizar el procesamiento de la información, ya que éste tiene una única tarea asignada, dándonos además un mayor nivel de seguridad gracias a la portabilidad y posicionamiento en la red del sistema embebido.

Agradecimientos

A mis padres, por haberme brindado la oportunidad de superarme y alcanzar una meta más. A ellos quienes con su apoyo, comprensión y cariño, me dieron las herramientas necesarias para poder dar un paso bastante importante en mi vida. Gracias por sus sacrificios e incontables esfuerzos, gracias por los consejos y ejemplos de lucha, y sobre todo, gracias por el amor incondicional que siempre me han brindado.

Gracias Papás.

Al profesor Arturo Zúñiga López, por brindarme el conocimiento y experiencia necesarios en la elaboración de este proyecto. Gracias por su apoyo y supervisión.

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Con el rápido incremento de los avances en las redes de computadoras, éstas nos ofrecen cada vez más beneficios como lo son, el obtener información de una forma casi inmediata, almacenar información en la nube, hacer compras en línea, etc. Sin embargo al utilizar la red no estamos del todo seguros, ya que existen personas que desean tener acceso a ella para obtener información o recursos para hacer mal uso de ellos. Por ello es importante contar con elementos de seguridad en las redes.

En la actualidad las compañías almacenan su información en bases de datos que están en alguno de sus servidores en su red y han tenido que abrir el acceso a dicha información para que los empleados puedan conectarse a la intranet, lo que las hace vulnerables a los ataques de los intrusos. Dichos ataques pueden ser evadidos utilizando por ejemplo un *Firewall* que bloquea el acceso no autorizado a la red o una VPN (*Virtual Private Network*, Red Virtual Privada) que permite una extensión segura de la red local a la red pública, pero debido a que los atacantes han evolucionado sus técnicas de intrusión, es necesario contar con herramientas adicionales que permitan elevar el nivel de seguridad y detectar accesos no autorizados en la red, una de estas herramientas es el IDS (*Intrusion Detection System,* Sistema de Detección de Intrusos), que bien se puede implementar en un sólo equipo HIDS (*Host IDS,* Sistema de Detección de Intrusos en Host) o en toda la red NIDS (*Network IDS*, Sistema de Detección de Intrusos de Red), siendo este ultimo el más eficiente ya que hace uso de algoritmos que permiten analizar el comportamiento de la red bajo un ataque y reconocer patrones que ayudan a detectar dichos ataques.

La implementación más común de un NIDS es a través de un servidor, pero tiene ciertas desventajas que impactan en el rendimiento y la velocidad de procesamiento debido a las múltiples funciones que el servidor realiza en la red. Es por ello que se busca implementar un NIDS en un sistema embebido que contrarreste dichas desventajas y sirva como complemento del servidor y no como una carga más a su funcionamiento, es decir se busca tener una solución en hardware (como un *firewall*) que simplemente están presentes como una **caja**

negra en la red, la cual provee seguridad y altos beneficios de rendimiento.

La detección de intrusiones permite a las organizaciones proteger sus sistemas de las amenazas que aparecen al utilizar una conectividad en red y la dependencia que tenemos hacia los sistemas de información.

Las principales razones para usar un NIDS son las siguientes:

- Prevenir problemas al disuadir a individuos hostiles
- Detectar ataques y otras violaciones de seguridad que no son prevenidas por otras medidas de protección
- Detectar preámbulos de ataques (normalmente pruebas de red y otras actividades)
- Documentar el riesgo de la organización
- Proveer información útil sobre las intrusiones que se están produciendo

Una ventaja de diseñar e implementar un NIDS en un sistema embebido es que este generalmente está orientado a minimizar los costos y maximizar la confiabilidad además operan en un ambiente dedicado con condiciones operacionales y escenarios muy específicos, lo que hace que sea sumamente importante que dichas condiciones y amenazas sean tomadas en cuenta cuando se diseñan las funciones de seguridad, que es la parte esencial que un NIDS tiene la tarea de proteger.

1.1. Objetivo General

- Implementar un NIDS en un sistema embebido que permita detectar ataques en una red, utilizando el algoritmo de reconocimiento de patrones Naive Bayes.

1.2. Objetivos Particulares

- Generar ataques en la red, para determinar cuales son los patrones que generan dichos ataques.
- Implementar un *Sniffer* utilizando la aplicación TCPDUMP, en el sistema embebido.
- Aplicar un pre-procesamiento a la información para generar formatos adecuados de los datos.
- Implementar el algoritmo *Naive Bayes*.
- Evaluar el desempeño del NIDS de acuerdo al índice de detección de intrusiones.

- Aplicar un post-procesamiento a la información para generar reportes de acuerdo a la información obtenida por el NIDS.

Capítulo 2

Marco Teórico

2.1. IDS, *Intrusion Detection System* (Sistema de Detección de Intrusos)

El objetivo principal del componente IDS es detectar, y posiblemente prevenir, y reconocer, ataques DoS[1] y ataques de acceso no autorizados. Para entender los diferentes tipos de ataques de red a los que una empresa se enfrenta, es necesario un profundo conocimiento de los diferentes tipos de tráfico que circula por la red, así como la intención de este tráfico.

La mayoría del tráfico que entra o que atraviesan sus redes tiene un propósito válido: para acceder a páginas web con HTTP, resolver el nombre a direcciones con DNS, enviar correo electrónico con SMTP, etc. Sin embargo, un pequeño porcentaje de tráfico tiene intenciones maliciosas. En estos casos, un Hacker podría realizar la ejecución de un ataque de reconocimiento, para determinar qué tipo de recursos están disponibles en la red, y luego podría ejecutar un ataque DoS para afectar su nivel de servicio o realizar un ataque no autorizado y abrir una puerta trasera en su red. Una solución IDS debe ser capaz de detectar este tipo de amenazas.

Los IDS caen bajo alguna de las tres estrategias siguientes:

- **Basado en Anomalías**
- **Basado en Firmas**
- **Híbrido**

Los sistemas Basados en Anomalías, capturan tráfico durante un periodo de tiempo y usan esto como una referencia para lo que es válido. Estos sistemas después comparan el

[1] Ver sección 2.2.1

nuevo tráfico con lo que es considerado "normal" y buscan anomalías. Una desventaja de estos sistemas es que tienden a generar demasiados falsos positivos (no son realmente ataques) esto es porque los patrones de tráfico cambian.

Los sistemas Basados en Firmas, comparan el tráfico con las firmas en busca de ataques. Una firma es una definición estática para examinar un paquete o paquetes; esto puede incluir información de cabeceras así como datos. Estos sistemas tienen muchas menos alarmas de falsos positivos. Sin embargo la principal desventaja es que no pueden detectar nuevos tipos de ataques a menos que se mantenga actualizada la base de firmas.

En muchas de las soluciones IDS de hoy, se utiliza un **enfoque Híbrido**, con ambas firmas y detección de anomalías conjuntamente para proporcionar la mejor detección de intrusos posible.

Los IDS son de dos tipos:

- **IDS basado en Red, NIDS**
- **IDS basado en Host, HIDS**

Un NIDS, *Network Intrusion Detection System*, es una analizador de protocolos, se conecta en la red en puntos clave y analiza el tráfico. Un NIDS puede ser usado para detectar ataques contra muchos dispositivos diferentes. Un buen NIDS debe tener la capacidad, cuando un ataque es detectado, para acceder al *Firewall* y configurar temporalmente un filtro para bloquear el tráfico malicioso. Esta es una herramienta excelente para cerrar el acceso de los Hackers incluso en áreas públicas de la red [9].

La gran mayoría de NIDS opera casi al igual que un *Sniffer*[2]. El dispositivo se encuentra en una red con sus interfaces en modo promiscuo, en busca de tráfico sospechoso. Cuando un paquete o conjunto de paquetes, coincide con una firma configurada, se genera una alarma[3] en la consola de administración [4].

Los HIDS, *Host Intrusion Detection System*, son software IDS corriendo en un host, como una PC o un servidor de archivos, que detectan ataques solamente contra ese host. Esto puede proporcionar una medida adicional de protección para los servidores críticos que no necesariamente están protegidos por un *Firewall* o un IDS. Una desventaja de los HIDS, es que requieren de un poder de procesamiento extra para examinar la información de los paquetes enviada al host [9].

[2]Un *Sniffer* es un analizador de protocolos de red.

[3]Un NIDS puede generar mucha información de registro (*Logs*). Esta información se debe revisar y analizar con cuidado diariamente. Al hacer esto se ganará una excelente comprensión de los patrones de tráfico en su red, así como lo que los Hackers están actualmente haciendo.

2.2. Ataques

En la actualidad es muy común encontrarse con ataques de red que surgieron a principios de los 90, y el hecho de que perduren es debido a que su finalidad se sigue cumpliendo en muchos sistemas actuales. Los ataques en una red están clasificados de distintas maneras según sea el objetivo que se persiga o el ambiente en el que se encuentre el atacante o la víctima del ataque.

2.2.1. Ataques DoS

Los ataques **DoS, *Denial Of Service*** están diseñados para interrumpir el servicio a un solo sistema o a una red completa. Un atacante usa este tipo de ataque para sobrecargar o utilizar de manera excesiva los recursos de un sistema o una red. Los ataques DoS pueden usar los dispositivos de red para enviar paquetes. Ellos también pueden forzar a las aplicaciones para que dejen de funcionar correctamente. Con una ataque DoS, el objetivo del atacante es evitar que los usuarios utilicen los servicios del sistema o la red. Estos ataques tienen como objetivo principal **el ancho de banda o la conectividad** de un sistema o una red[3].

Consumo de Ancho de Banda

En los ataques de ancho de banda, la red es inundada con una gran cantidad de paquetes y esto conduce al agotamiento de los recursos de red disponibles (Ancho de banda, conectividad, etc.), de tal forma que usuarios legítimos no pueden acceder a ellos[1]. Entre los principales ataques de este tipo se encuentran:

- **SMURF *Attack*:** Este ataque se basa en mandar un gran número de peticiones *Echo* (ICMP) a direcciones de *Broadcast* usando una IP de origen falsa. Esto provoca que la IP de origen sea inundada con multitud de respuestas.
- **ICMP *Flood*:** En este ataque se inunda a la víctima con paquetes ICMP *Echo Request*.
- ***Fraggle Attack*:** Es similar al ataque *Smurf* pero en este caso se envía tráfico UDP en lugar de ICMP.

Ataques a la Conectividad

En los ataques de conectividad, una computadora es inundada por un gran volumen de solicitudes de conexión que conduce a un agotamiento de todos los recursos del sistema operativo, por tanto, la computadora no es capaz de procesar las solicitudes de usuarios legítimos[1].

Entre los principales ataques de este tipo se encuentran:

- **SYN *Flood Attack*:** Consiste en enviar muchos paquetes TCP/SYN con la dirección de origen falseada. Esto provoca que el servidor espere las respuestas que nunca llegan, provocando un consumo elevado de recursos que afectan al rendimiento del servidor.

2.2.2. ARP *Spoofing*

El ataque ARP *Spoofing* se trata de la construcción de tramas de solicitud y respuesta ARP modificadas, con el objetivo de modificar la tabla ARP (Relación IP-MAC) de una víctima y forzarla a que envíe los paquetes a un *host* atacante en lugar de hacerlo a su destino legítimo.

El ataque puede ser también de la siguiente manera, el atacante envía tramas ARP modificadas en *Broadcast* o a un *host* en especial anunciando que la dirección MAC del *Gateway* legítimo por ejemplo ha cambiado a la dirección MAC del atacante. Una vez que la máquina de la víctima actualiza su caché de ARP, todas las peticiones que son para el *Gateway* legítimo pasan a la máquina del atacante, donde los paquetes pueden ser modificados, leídos o retirados [4].

Funcionamiento de ARP y su relación con IP

Las capas de red utilizan direcciones que identifican a emisor y receptor. En TCP/IP, si el nivel 2 (*Datalink Layer*) es *Ethernet*, las direcciones son direcciones MAC, que son números de 48 bits. En el nivel 3, en TCP/IP, se utiliza IP, que tiene direcciones de 32 bits.

Cuando una máquina (origen) requiere enviar un paquete a otra máquina (destino) examina su tabla de *Routing* (ruteo). Si la dirección IP de destino está en un segmento conectado directamente a la máquina origen, entonces, la máquina origen sabe que puede entregar el paquete "localmente". Por el contrario, si la máquina de destino no está conectada a un segmento local, la máquina origen debe entregar el paquete a un Router, el cual será capaz de hacer llegar el paquete a su destino.

En ambos casos la máquina origen debe conocer la dirección MAC de destino, aunque no tiene porqué saberla *a priori*.

El protocolo encargado de averiguar la dirección MAC asociada a una dirección IP conocida se llama **ARP (*Address Resolution Protocol*, Protocolo de Resolución de Direcciones)**. Los Switches, no necesitan usar ARP para aprender las direcciones MAC de origen que llegan por sus puertos. Los Switches elaboran una lista (*mas-address-table*) en la que aparecen las direcciones MAC que tienen accesibles por cada uno de sus puertos. Así, cuando el Switch deba entregar una trama lo hará solamente por el puerto en el que se encuentre la MAC de destino.

El funcionamiento de ARP es muy sencillo. Cuando un nodo necesita averiguar la dirección MAC de una determinada IP, el nodo envía una petición "ARP *who has*" preguntando a todos los nodos "¿Cuál es la MAC de esta IP?". Esta petición lleva como dirección MAC de origen la del nodo que hace la petición, y como dirección MAC de destino la MAC de *Broadcast*. Por tanto, las peticiones de "ARP *who has*" las reciben todos los nodos. Aquel que tenga la IP por la cual se pregunta en la petición de ARP *who has* debe contestar con un paquete "ARP *is at*", indicando su dirección MAC. La respuesta de ARP *is at* tiene como dirección MAC de origen la del nodo que está contestando, y como la dirección MAC de destino del nodo que hizo la pregunta [2].

2.2.3. DHCP *Spoofing*

Este ataque consiste básicamente en falsificar paquetes DHCP, instalando un servidor DHCP falso o un software que emule las funciones del mismo de tal forma que responda a peticiones DHCPDISCOVER de los clientes[6].

Comunicación entre el cliente y el servidor

La comunicación entre el Cliente y el Servidor DHCP se muestra en la figura 2.1 y se explica a continuación.

- Cuando un equipo se conecta a la red y solicita una dirección IP envía un paquete **DHCPDISCOVER** a la dirección de *Broadcast* (UDP) esperando respuesta por algún servidor DHCP.
- Éste contestara a tal petición enviando un paquete *Unicast* denominado **DHCPOFFER**, que contiene varios parámetros de configuración como IP, *Gateway* etc.
- En este punto, el cliente puede recibir paquetes DHCPOFFER de varios servidores DHCP y para elegir a algún servidor, el cliente se basa en lo siguiente:
  - Si la oferta propuesta corresponde a una dirección previamente asignada (ya que son recordadas por el cliente), el cliente seleccionará ésta.
  - En caso de que la propuesta no esté relacionada con una dirección IP previa, el cliente adquirirá la primera oferta recibida.
- Como respuesta a la oferta recibida el cliente enviará un paquete **DHCPREQUEST** a la dirección *Broadcast* pidiendo autorización para utilizar esa configuración a lo que el servidor responderá, o bien con un paquete *Unicast* **DHCPACK** autorizando el uso de dicha configuración, o bien con un **DHCPNAK** negando el uso de tales parámetros[6].

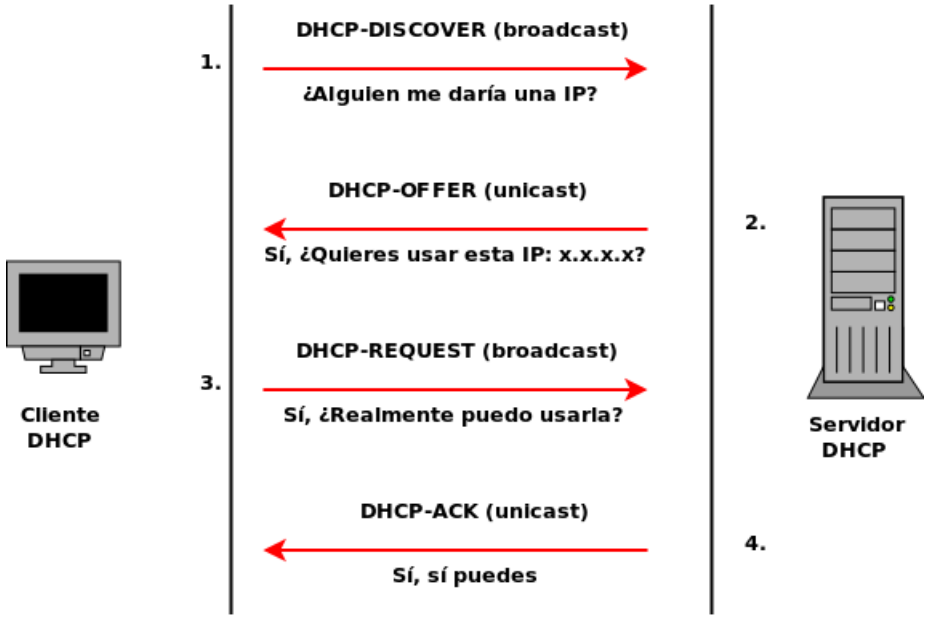


Figura 2.1: Negociación DHCP

Cuando se logra hacer un ataque DHCP *Spoofing*, el atacante puede configurar su equipo de tal manera que envíe y reciba información del cliente y del *Gateway* legítimo, y de esta manera lograr un ataque *Man In The Middle* (Hombre en el medio).

2.2.4. *Man In The Middle*

Un ataque **MITM, *Man In The Middle*** es aquel donde el atacante esta en control activo de la más alta capa de conversación entre dos víctimas. Generalmente es la capa 7, aunque los ataques MITM contra cifrado pueden ocurrir en capa 3 mientras se usa *IPsec*. El clásico ejemplo de MITM (Ver figura 2.2) es un cliente que se comunica con su banco en la red. El cliente piensa que está hablando con el cajero del banco y el banco piensa que está hablando con el cliente. En realidad ambas conversaciones están dirigidas hacia el atacante, quien esta modificando datos tales como, números de cuenta, montos de transferencia, etc[4].

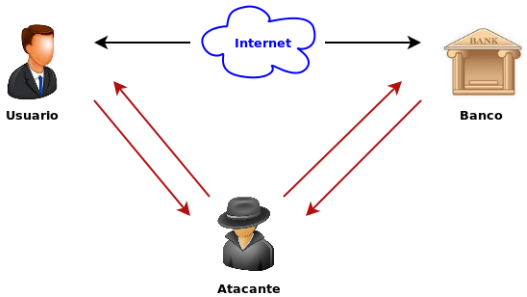


Figura 2.2: Esquema del ataque MITM

2.2.5. TCP SYN *Flood*

Generalmente este tipo de ataque es llamado *Flood* porque el ancho de banda que se tiene para la comunicación es inundado con una cantidad de paquetes a una frecuencia y tamaño considerables, lo que hace que el sistema se quede sin recursos para las demás tareas que realiza y la tarjeta o tarjetas de red se saturan y dejan de funcionar correctamente hasta el grado de detener su funcionamiento por completo.

Los ataques TCP SYN *Flood* están diseñados para tomar ventaja de la metodología utilizada en el establecimiento de una nueva conexión TCP, referidos como TCP de tres vías (*Three-Way Handshake*). En la figura siguiente (fig. 2.3) se ilustra como una conexión TCP es establecida.

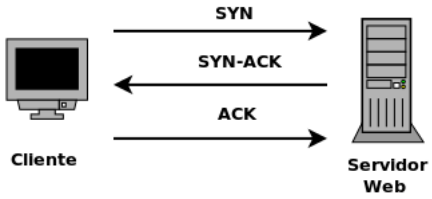


Figura 2.3: Establecimiento de la conexión TCP

El cliente trata de establecer una conexión con el servidor web. Primero envía un paquete **SYN (sincronizar)** al servidor para sincronizar los números de secuencia. Estipula su número de secuencia inicial **(ISN)**. Para inicializar la conexión, el cliente y el servidor deben sincronizar el número de la otra secuencia. El campo de reconocimiento (***Acknowledgment* ACK)** es puesto en 0 porque este es el primer paquete de la conexión de tres vias y no hay acuses de recibido hasta ahora. En el segundo paquete el servidor envía un acuse de recibo y su propio **SYN (SYN-ACK)** de vuelta al cliente. El servidor reconoce la solicitud del cliente, pero también envía su propia solicitud de sincronización. El servidor incrementa el número de secuencia del cliente por uno y, además, la utiliza como el número de acuse de recibo. Para finalizar la conexión, el cliente envía un paquete (ACK) de acuse de recibido al servidor web. El cliente utiliza la misma metodología usada por el servidor para proporcionar un número de confirmación.

En los ataques **TCP SYN *Flood***, el atacante genera paquetes falsos para aparentar una nueva petición de conexión válida. Estos paquetes son recibidos por el servidor, pero la conexión nunca se completa. Por otra parte, el servidor trata de responder sin completar con éxito las conexiones. Después de que varios de estos paquetes son enviados al servidor, el servidor puede dejar de responder a las nuevas conexiones hasta que sus recursos están disponibles para procesar las solicitudes adicionales o cuando el ataque se detiene. La figura 2.4 muestra como trabaja un ataque SYN *Flood*. El atacante envía numerosos paquetes SYN falsos al servidor web [3].

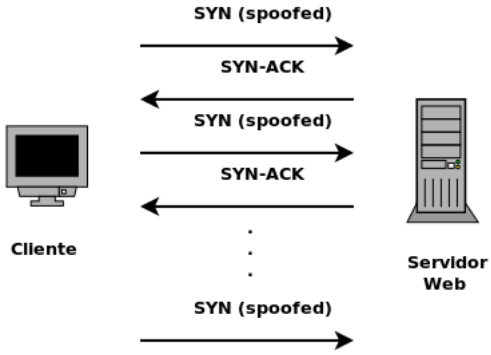


Figura 2.4: Esquema TCP SYN *Flood*

2.3. Algoritmos

El análisis de una gran cantidad información o resultados obtenidos bajo alguna circunstancia de experimentación, es muy importante ya que nos otorga un panorama muy amplio de posibles comportamientos futuros, permitiendo así, tomar decisiones o predecir cualquier tipo de riesgo o resultado que pueda ocurrir. Existen dos tipos de algoritmos de predicción que realizan éste análisis requerido[10]:

- **Regresión:** El objetivo de este tipo de análisis es determinar, de acuerdo a un resultado dado, el valor de los parámetros que produjeron ese resultado. Por ejemplo, se ha reportado el uso de métodos de regresión para asegurar la calidad de los sistemas de cómputo mediante el análisis de los errores de ejecución.

- **Clasificación:** La clasificación trata de encontrar las características que identifican a un grupo para ser clasificado dentro de cierta clase. Este conocimiento puede ser utilizado para entender el comportamiento del sistema que generó los datos y de esta forma predecir la clase a que pertenecerá una nueva instancia. Entre los algoritmos de clasificación se encuentran:

  - **Análisis discriminante**
  - **K-Vecinos más cercanos**
  - **Redes neuronales**
  - **Árboles de decisión**
  - **Vectores Soporte**
  - **Naive Bayes**

2.3.1. Naive Bayes

Aprendizaje Bayesiano

El Aprendizaje Bayesiano proporciona un método probabilístico para interactuar. Es un método importante no sólo porque ofrece un análisis cualitativo de los atributos y valores que pueden intervenir en un problema, sino porque da cuenta también de la importancia cuantitativa de esos atributos. En el aspecto cualitativo podemos representar cómo se relacionan esos atributos ya sea en una forma causal, o señalando simplemente la correlación que existe entre esas variables (o atributos). Cuantitativamente, da una medida probabilística de la importancia de esas variables en el problema (y por lo tanto una probabilidad explícita de las hipótesis que se formulan) [8].

Clasificación de Patrones

Cualquier sistema de clasificación de patrones se basa en lo siguiente: dado un conjunto de datos (divididos en dos conjuntos de entrenamiento y de test) representados por pares <atributo, valor>, el problema consiste en encontrar una función f(x) (llamada hipótesis) que clasifique dichos ejemplos. La idea de usar el Teorema de Bayes en cualquier problema de aprendizaje (en especial de clasificación) es que podemos estimar las probabilidades a posteriori de cualquier hipótesis consistente con el conjunto de datos de entrenamiento para así escoger la hipótesis más probable. Para estimar estas probabilidades se han propuesto numerosos algoritmos, entre los que cabe destacar al algoritmo Naive Bayes [8].

Clasificador Naive Bayes

Uno de los métodos de aprendizaje bayesiano eminentemente práctico es el aprendizaje Naive Bayes, a menudo llamado el clasificador Naive Bayes (clasificador ingenuo de Bayes). En algunos dominios se ha demostrado que su rendimiento es comparable al del aprendizaje de redes neuronales y árboles de decisión. El clasificador Naive Bayes se aplica a las tareas de aprendizaje, donde cada instancia x se describen por un conjunto de valores de atributos y donde la función objetivo *f(x)* puede adoptar cualquier valor entre un conjunto finito V.

Dado un conjunto de ejemplos de entrenamiento con valores de atributo $< a_1, a_2 \ldots a_n >$, el clasificador Naive Bayes se basa en encontrar la hipótesis más probable que describa a ese ejemplo, el enfoque para clasificar esa nueva instancia es asignar el valor objetivo más probable, v_{MAP}, dados los valores de los atributos $< a_1, a_2 \ldots a_n >$ que describen la instancia.

$$v_{MAP} = \underset{v_j \in V}{argmax}\, P(v_j | a_1, a_2, \ldots a_n) \tag{2.1}$$

Es decir la probabilidad de que conocidos los valores que describen a ese ejemplo, pertenezcan a la clase v_j (donde v_j es el valor de la función de clasificación $f(x)$ en el conjunto finito V). Podemos usar el Teorema de Bayes para reescribir esta expresión como:

$$v_{MAP} = \underset{v_j \in V}{argmax} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)}$$

$$= \underset{v_j \in V}{argmax} P(a_1, a_2 \dots a_n | v_j) P(v_j) \tag{2.2}$$

Podemos estimar $P(v_j)$ contando las veces que aparece el ejemplo v_j en el conjunto de entrenamiento y dividiéndolo por el número total de ejemplos que forman este conjunto. Para estimar el término $P(a_1, \dots a_n | v_j)$, es decir, las veces en que para cada categoría aparecen los valores del ejemplo x, se debe recorrer todo el conjunto de entrenamiento. Éste cálculo resulta impracticable para un número suficientemente grande de ejemplos por lo que se hace necesario simplificar la expresión. Para ello se recurre a la hipótesis de independencia condicional con el objeto de poder factorizar la probabilidad. Esta hipótesis dice lo siguiente: *Los valores a_j que describen un atributo de un ejemplo cualquiera x son independientes entre sí conocido el valor de la categoría a la que pertenecen.* Así la probabilidad de observar la conjunción de atributos a_j dada una categoría a la que pertenecen es justamente el producto de las probabilidades de cada valor por separado: $P(a_1, a_2 \dots a_n | v_j) = \prod_i P(a_i | v_j)$. Sustituyendo esta ecuación en 2.2, tenemos el enfoque usado por el clasificador Naive Bayes.

Clasificador Naive Bayes:

$$v_{NB} = \underset{v_j \in V}{argmax} P(v_j) \prod_i P(a_i | v_j) \tag{2.3}$$

Donde v_{NB} denota la salida del valor objetivo en el clasificador Naive Bayes [8].

2.4. Sistema Embebido

Un **Sistema Embebido** es un sistema de computadora de propósito especial que está diseñado para llevar acabo un muy pequeño conjunto de actividades asignadas. El primer sistema embebido reconocido fue el *Apollo Guidance Computer* desarrollado por Charles Draper y su equipo. Hoy en día son ampliamente utilizados para servir a diversos fines; Algunos ejemplos son los siguientes:

- Equipos de red, tales como *Firewalls, Routers, Switches*, etc.
- Aparatos electrónicos de consumo, tales como reproductores de MP3, teléfonos móviles, PDAs, cámaras digitales, cámaras de vídeo, sistemas de entretenimiento en el hogar, etc.
- Aparatos electrodomésticos como microondas, lavadoras, televisores, etc.

- Los sistemas de misión crítica, tales como satélites y de control de vuelo.

Los siguientes son los factores clave que diferencian a un sistema embebido de una computadora de escritorio.

- La mayoría de los sistemas embebidos tienen restricciones de tiempo real.
- Hay multitud de arquitecturas de CPU (como ARM®, MIPS®, PowerPC™, etc) que se utilizan en sistemas embebidos. Los sistemas embebidos utilizan los procesadores de aplicaciones específicas. Por ejemplo, el procesador de la cámara digital está especialmente adaptado para la captura de imágenes y la renderización.
- Los sistemas embebidos tienen (y necesitan) muy pocos recursos en términos de RAM, ROM u otros dispositivos de E/S, en comparación con una computadora de escritorio.
- La administración de energía es un aspecto importante en la mayoría de los sistemas embebidos.
- Un sistema embebido es diseñado desde el punto de vista del hardware y el software, teniendo en cuenta una aplicación específica o un conjunto de aplicaciones. Por ejemplo, su reproductor de MP3 puede tener un hardware decodificador de MP3 independiente construido en su interior.

Con el paso del tiempo, los sistemas embebidos más complejos comenzaron a surgir y junto con ello una creciente lista de características que un sistema embebido debe apoyar. Todos estos requisitos obligatorios hacen uso de un sistema operativo en sistemas embebidos que por lo menos deben proporcionar procesos multitarea/multihilo y de gestión de la comunicación entre procesos de memoria, temporizadores, etc[7].

2.4.1. PICO820 series

El sistema embebido que se utilizará para este proyecto es el **PICO820 Series Intel® ATOM™All-In-One** (Ver figura 2.5) e integra el chipset Intel System Controller Hub US15W que ofrece un rendimiento notable en el sistema a través de interfaces de alto ancho de banda, múltiples funciones de E/S para aplicaciones interactivas y diversas soluciones informáticas embebidas, a continuación se especifican algunas otras características principales de este sistema [5].

- **CPU**
  - **procesador Intel®ATOM™Serie Z500.**
- **Memoria del sistema**
  - Una 200-pin DDR2 sin búfer socket SODIMM.

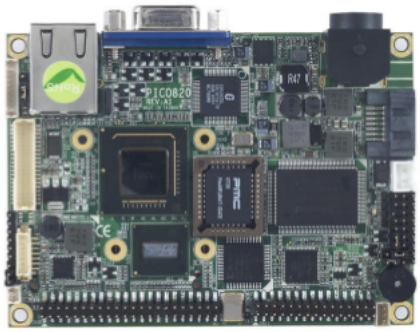


Figura 2.5: Sistema Embebido PICO820 Z500

- Máximo de memoria 2 GB DDR2 400/533 MHz.

- **Ethernet**
  - Un puerto con RTL8111B para Gigabit/Fast Ethernet.
  - Un conector RJ-45.

Para poder utilizar el sistema embebido hubo que adaptarle los componentes necesarios que necesitaba, los cuales fueron:

- Alimentación de corriente.
- 2 Puertos USB.
- Botones de encendido y reset.
- Led de encendido y uso de disco duro.
- Disco duro y memoria RAM.
- Base de acrílico para todo el sistema ya integrado.
- Instalación de sistema operativo Kali Linux.

Capítulo 3

Desarrollo

3.1. Implementación de Ataques

Se realizará la implementación de los ataques **TCP SYN *Flood* y DHCP *Spoofing*,** para los cuales se utilizarán las herramientas ***Hping3***[1] **y *Ettercap***[2]**,** de las cuales se explicará su funcionamiento conforme se mencionen en las implementaciones. La topología de red (ver fig. 3.1) y la tabla de direccionamiento (ver tabla 3.1) que se utilizarán en los ataques serán las mismas y se muestran a continuación.

Tabla 3.1: Tabla de Direccionamiento

Host	Interfaz	IP	MAC	Gateway
PC1	NIC	192.168.1.10	b8:ac:6f:b4:ec:99	192.168.1.253
PC2		192.168.1.20	b8:ac:6f:b4:a0:80	192.168.1.253
PC3		192.168.1.30	b8:ac:6f:b5:52:e7	192.168.1.253
PC4		192.168.1.40	b8:ac:6f:b5:52:f3	192.168.1.253
NIDS	NIC	192.168.1.254	00:60:e0:55:9a:5d	192.168.1.253
Switch	VLAN	192.168.1.1	-	-
Router	fa0/0	192.168.1.253	00:23:33:23:b3:20	-
	fa0/1	DHCP	00:23:33:23:b3:21	-

[1]Ver apéndice A.2 para instalación y más información
[2]Ver apéndice A.3 para instalación y más información

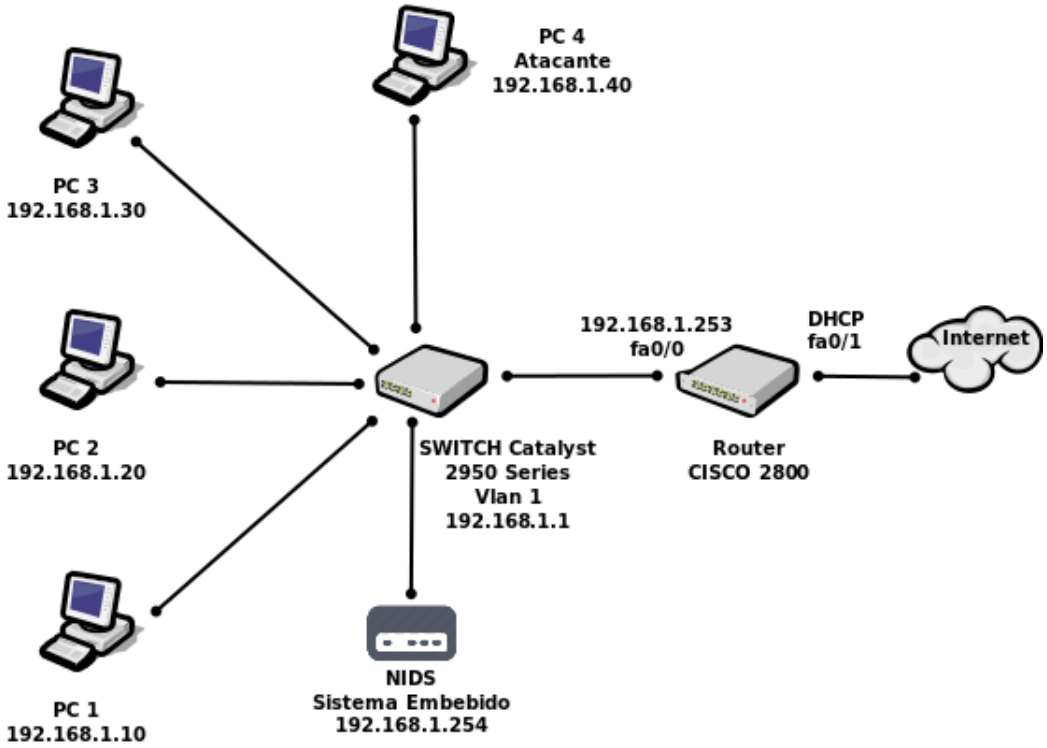


Figura 3.1: Topología de Red

3.1.1. SYN *Flood*

Usaremos la herramienta ***Hping3*** para realizar el ataque y nos basaremos en la topología de red de la figura 3.1 y la tabla de direccionamiento 3.1.

Para poder entender de mejor manera como se generará el ataque y cuál será la metodología que se utilizaŕa, se necesita conocer el funcionamiento de esta herramienta, lo cual se explica a continuación con un ejemplo:

```
Atacante~#hping3 -I eth0 -p 80 --flood -S --rand-source 192.168.1.1
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

[**-I]** especifica qué interfaz de red se va a utilizar, en este caso `eth0`.

[**- p]** especifica el puerto al cual se lanzará el ataque, puerto 80 para este ejemplo

[**- -flood]** envía paquetes lo más rápido posible, sin tener en cuenta las respuestas entrantes.

[**- S]** coloca la bandera SYN TCP

[**- -rand-source]** se utiliza para establecer una conexión con múltiples direcciones IP fuente aleatorias, en este ejemplo el destino (la víctima) es la IP `192.168.1.1`

La opción **[- -rand-source]**, nos da una ventaja además de crear direcciones IP aleatorias de origen, y es que para la detección de este ataque, nos apoyaremos en el protocolo **ARP**, el cual es utilizado por el Switch, cuando quiere conocer las direcciones IP de origen, utiliza el protocolo **ARP** y lo manda en *Broadcast*, lo que nos permite ver este tráfico en el NIDS.

Es necesario generar direcciones IP fuente diferentes, para que el destino trate de hacer una conexión para cada una de ellas y así dejarlo esperando el ACK que confirme la conexión, esto sirve no solo para consumir en ancho de banda de la NIC si no también para consumir recursos del CPU obligándolo a que ejecute más procesos.

Generando el ataque

En este caso el ataque será dirigido a la PC2 y el procedimiento para generar el ataque es el siguiente:

- Se abre una consola y como usuario `root` tecleamos la siguiente instrucción:

```
Atacante~#hping3 -I eth0 -p 80 --flood -S --rand-source 192.168.1.20
HPING 192.168.1.20 (eth0 192.168.1.20): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

- Dejamos que el ataque continúe por cierto tiempo, hasta que se envíen arriba de un millón de paquetes para que la denegación de servicio se vea reflejada en la víctima, en este caso la PC2.

- Podemos ejecutar un analizador de paquetes (*TCPDUMP*[3] por ejemplo) mientras el ataque está en ejecución para saber cuantos paquetes han sido enviados, pero la cantidad de paquetes que tiene mostrar es muy elevada y dependiendo de la capacidad del ordenador en donde se ejecute éste puede o no, dejar de tener un correcto funcionamiento.

3.1.2. DHCP *Spoofing*

Para generar este ataque usaremos la herramienta *Ettercap*, y nos basaremos en la topología de red de la figura 3.1 y la tabla de direccionamiento 3.1.

Como ya se explicó anteriormente, el objetivo de este ataque es tener un servidor DHCP intruso en la red, éste suplantará al auténtico servidor DHCP que en este caso es el Router. Una vez que el ataque es efectuado, la PC que solicite DHCP recibirá los OFFER del Router y del intruso, y en caso de ser el intruso el que otorgue los parámetros, la PC tendrá como *Gateway* al intruso. En este punto, se puede realizar un ataque MITM, sólo se necesita que

[3] Ver apéndice A.5 para más información

el intruso redirija el tráfico al *Gateway* original, y la información que éste le mande, se la enviará de nuevo a la PC.

Generando el ataque

Para ejecutar *Ettercap* se abre una terminal, se inicia como root y se introduce la siguiente instrucción:

```
Intruso~#ettercap -G
ettercap 0.8.0 copyright 2001-2013 Ettercap Development Team
```

Una vez hecho esto, se abrirá *Ettercap* en modo gráfico y se configurarán los siguientes parámetros para comenzar con el ataque:

1. Se elige la tarjeta de red que se utilizará (Figura 3.2), en este caso `eth0`: **Sniff>Unified Sniffing>eth0**

Figura 3.2: Selección de interfaz en *Ettercap*

2. Para poder empezar con el ataque se debe iniciar lo siguiente: **Start>Start Sniffing**

3. Después se elige el ataque **MITM>Dhcp spoofing** (Figura 3.3) y colocamos los siguientes parámetros:

   - IP Pool: 192.168.1.100 - 192.168.1.150
   - Netmask: 255.255.255.0
   - DNS Server IP: 192.168.1.40

   La opción **IP Pool** es opcional y el **DNS Server** puede ser la IP original o bien la IP del intruso de cualquier manera el que resolverá ese parámetro será el verdadero servidor cuando reenviemos la información hacia el.

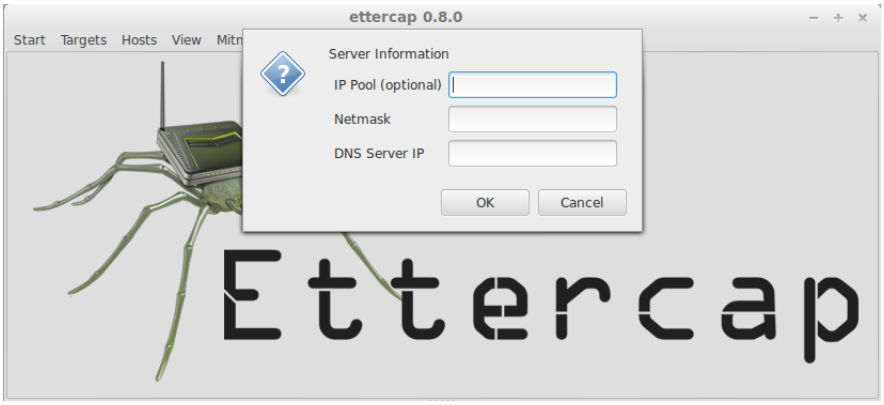


Figura 3.3: Selección del ataque MITM en *Ettercap*

Para conseguir que el ataque funcione correctamente, es decir, que el flujo de información entre la PC y el *Gateway* verdadero se realice normalmente, se necesita activar el ***Forwarding*** en el intruso, cuando el IP *Forwarding* está activo, la PC intenta entregar los paquetes que no están destinados a ella (es un comportamiento similar al de un Router), de esta manera la comunicación no se vera afectada.

Se comprueba el estado de IP *Forwarding*:

```
Intruso~#cat /proc/sys/net/ipv4/ip_forward
0
Intruso~#
```

Se observa que está desactivado, se procede entonces a activarlo para permitir el reenvío de paquetes:

```
Intruso~#echo 1 > /proc/sys/net/ipv4/ip_forward
Intruso~#cat /proc/sys/net/ipv4/ip_forward
1
Intruso~#
```

De esta manera el ataque estará completo, y seremos capaces de recibir todo el tráfico que circule entre la PC y el *Gateway* verdadero.

3.2. Implementación de Naive Bayes

El algoritmo de Naive Bayes explicado anteriormente, funciona en base a probabilidades, y para entenderlo mejor, se explicará sobre la marcha su funcionamiento, resolviendo el problema de detectar los ataques **DHCP *Spoofing* y TCP SYN *Flood***.

El algoritmo funciona en base a valores de atributos que intervienen en un problema, con estos atributos se conforma un conjunto de ejemplos de entrenamiento, y ante una nueva instancia conformada por estos atributos, el algoritmo se basa en encontrar la hipótesis más probable que describa a ese ejemplo, en este caso se determinará si esa instancia es un ataque,

un posible ataque, un ataque desconocido o no es un ataque.

El conjunto de ejemplos de entrenamiento (base de firmas), es un conjunto de ejemplos que muestra el comportamiento de los posibles ataques mencionados, es decir, le estamos diciendo al algoritmo en base a ejemplos, si tal comportamiento es considerado o no como un ataque. Los atributos del conjunto de entrenamiento son los siguientes:

- **TCP SYN *Flood*: *Logs*[4] x Minuto**
  Cuando se genera este ataque, la PC atacada, comienza a enviar paquetes ARP en *Broadcast* con direcciones de origen de enlace local (169.254.x.x) en el caso de que no se tenga un *Gateway* y hacia la dirección del *Gateway* por defecto en caso contrario, esta presencia de paquetes en la red se registra en el archivo de *Logs*, y son estos registros los que se analizan para otorgar a este atributo el valor de **0** si no hay registro de más de 10 *Logs* x Minuto, y el valor de **1** si hay presencia de estos registros (más de 10 x min.). Para determinar el número de *Logs* se realizó un análisis del comportamiento de una PC bajo el ataque, y se obtuvo que el promedio de paquetes ARP que enviaba en *Broadcast* era de 40 x minuto y 4 x segundo, y para tener un mejor análisis y resultado del algoritmo, se determinó tras varias pruebas que el valor ideal de *Logs* era de 10 x minuto y 2 x segundo.

- **TCP SYN *Flood*: *Logs* x Segundo**
  Para asignarle un valor a este atributo, se realiza un procedimiento análogo al atributo anterior, pero en lugar de analizar la cantidad de *Logs* x minuto, se analizan la cantidad de *Logs* x segundo, teniendo el valor de **0** si no hay registros de más de 2 *Logs* x Segundo y el valor de **1** en caso contrario.

- **DHCP *Spoofing*: Match de MAC o IP**
  Un NIDS debe conocer datos *a priori* que le ayuden en su funcionamiento, en este caso, son los valores de la IP y MAC del servidor legítimo, y son estos datos los que se analizan en cada paquete para detectar si los paquetes OFFER o ACK se generaron por nuestro servidor o por el de un intruso, teniendo así el valor **0** si fue el servidor legítimo o el valor de **1** si es la IP o MAC de otro servidor.

- **Tipo de MAC**
  Cuando se duplica una IP o una MAC, generalmente se hace por medio de una herramienta de software (como *Hping3*), y es entonces cuando se generan direcciones MAC NO válidas, es decir, direcciones MAC que no pueden ser asignadas a una PC, este es el caso de una dirección MAC de tipo *Multicast*, donde el *bit* menos significativo del primer octeto de la dirección MAC, está puesto en 1 (en formato binario). Tener una dirección MAC de tipo *Multicast* asignada a una PC es algo anómalo en una red, lo que indica que se puede estar bajo un ataque, para dar valores a este atributo, se analiza el tipo de MAC que tiene la PC y se asigna el valor **0** si la MAC es de tipo *Unicast*, el valor de **1** si es de tipo *Multicast* y **2** si es *Broadcast*.

[4]Ir al Apéndice A.1 para ver su configuración

- **TCP SYN *Flood*: Banderas**
  Los paquetes enviados por este ataque en particular, se caracterizan por tener la bandera **SYN** activa, es por eso que se analizarán en especial las banderas **SYN, ACK, *Reset, Push* y *Fin***, para diferenciar entre los paquetes normales de conexión TCP y los de un ataque, y los valores de este atributo serán **0** si no hay presencia de banderas, **1,2 o 3** si están presentes las banderas **SYN, SYN-ACK o ACK** respectivamente y **4** si es alguna otra bandera o combinación de estas.

- **Ataque**
  En base a los valores de cada uno de los atributos anteriores, éste atributo toma el valor de **0** si se considera que NO se está bajo un ataque, el valor de **1** para un ataque (DHCP *Spoofing* o TCP SYN *Flood*) positivo, el valor de **2** para un posible ataque(DHCP o TCP) y **3** para un tipo de ataque desconocido.

Estos atributos nos darán un conocimiento del comportamiento esencial de un ataque, así al tener una nueva instancia con estos atributos, la probabilidad obtenida para cualquiera de las cuatro clases (Sí, No, Posible y Otro) tendrá un margen de error muy pequeño. El conjunto de ejemplos de acuerdo con los atributos anteriores se muestra en la tabla 3.2.

Tabla 3.2: Conjunto de ejemplos de aprendizaje

Logs ARP		**Match**	**Tipo de**	**Banderas**	**Ataque**
>10 x Min.	**>2 x Seg.**	**MAC o IP**	**MAC**	**TCP**	
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	2	0
0	0	0	0	3	0
0	0	0	0	4	0
1	0	0	0	0	3
1	0	0	0	1	1
1	0	0	0	2	1
1	0	0	0	3	1
1	0	0	0	4	2
1	1	0	0	0	3
1	1	0	0	1	1
1	1	0	0	2	2
1	1	0	0	3	2
1	1	0	0	4	3
1	1	1	0	0	1
1	1	0	1	0	3
1	1	0	2	0	2
0	0	1	0	0	1
0	0	0	1	0	0
0	0	0	2	0	0
1	0	1	0	0	1
1	0	0	1	0	2
1	0	0	2	0	3
0	1	1	0	0	1
0	1	0	1	0	2
0	1	0	2	0	3
0	1	0	0	0	2

Organizar los datos de manera práctica agiliza el análisis, la forma de organizar los datos es a través de un conteo en las instancias, realizado de la siguiente forma: para cada atributo, se cuenta el número de veces que aparece determinado valor junto con una clase en particular. Como ejemplo tomamos el caso del atributo **Tipo de MAC**, se cuenta el número de veces que aparece Tipo de MAC = 0 y Ataque = 0 que es igual a 6, y las veces que aparece Tipo de MAC = 0 y Ataque = 1 que es igual a 8 y así sucesivamente. El total de conteos para cada clase por atributo se presenta en cada tabla correspondiente (Tablas 3.3 a 3.8).

Tabla 3.3: Conteo de *Logs* >40 x minuto

>10xMin \ **Ataque**	**0**	**1**	**2**	**3**
0	7	2	2	1
1	1	6	5	4

Tabla 3.4: Conteo de *Logs* >2 x segundo

>2xSeg \ **Ataque**	**0**	**1**	**2**	**3**
0	7	5	2	5
1	1	3	5	3

Tabla 3.5: Conteo de Match MAC o IP

Match \ **Ataque**	**0**	**1**	**2**	**3**
0	8	4	7	5
1	0	4	0	0

Tabla 3.6: Conteo de Tipo de MAC

Tipo MAC \ **Ataque**	**0**	**1**	**2**	**3**
0	6	8	4	2
1	1	0	2	1
2	1	0	1	2

Tabla 3.7: Conteo de Banderas TCP

Banderas \ Ataque	**0**	**1**	**2**	**3**
0	4	4	4	4
1	1	2	0	0
2	1	1	1	0
3	1	1	1	0
4	1	0	1	1

Tabla 3.8: Conteo de Ataques

Ataque			
0	1	2	3
8	8	7	5

Ahora que tenemos la tabla de conteos para cada atributo, podemos calcular las tablas de probabilidades. Para calcular la probabilidad de cada uno de los atributos, se realiza lo siguiente: Tomaremos como ejemplo calcular la probabilidad de tener el caso **Match = 0 y Ataque = 0**, de acuerdo con la tabla de conteos, hay un total de 8 apariciones, por lo tanto la probabilidad sería **8** entre el número total de apariciones donde **Ataque = 0**, y esto sería igual a $8/8 = 1$.

Hasta ahora todo parece bien, pero consideremos el caso de calcular la probabilidad cuando **Match = 1 y Ataque = 0**, el total de apariciones de esta combinación de atributos es **0**, por lo tanto la probabilidad sería $0/8 = 0$, lo cual eliminaría la posibilidad de que se presente este caso si utilizáramos el mismo procedimiento que en el ejemplo anterior, pero en circunstancias reales, no se puede asegurar *a priori* que nunca se presentará un caso como este durante el análisis que realiza en NIDS.

Para eliminar este anterior problema, se utilizó el *estimador de Laplace*[16], el cual considera que cada valor del atributo V_i, es equiprobable respecto a todos los posibles valores de ese atributo, y que existe una constante μ tal que: $\mu = |v|$ ($|$v$|$ es el número posible de distintos valores que puede tomar dicho atributo). Considerando esto la probabilidad para cada atributo está dada por:

$$P[C_j|V_i] = \frac{CasosFavorables + \mu * P_i}{CasosTotales + \mu}; i \in \{Atributos\}, j \in \{TipoAtaque\} \tag{3.1}$$

Donde P_i es la probabilidad de v_i para todos los valores de c_y y v_i.

Ahora sí calculamos la probabilidad de la combinación **Match = 1 y Ataque = 0**, tenemos que $|v|$ son los posibles valores del atributo **Match**, y es igual a 2, se considera *a*

priori que los 2 valores de este atributo son igualmente probables, de modo que la probabilidad de Match = 1/2, y ahora utilizando la fórmula del estimador de *Laplace* se tiene que la probabilidad es $(0 + 2 * 1/2)/(8 + 2) = 0{,}1$.

De igual manera que en el caso anterior, se calculan todas las probabilidades de las combinaciones de cada atributo en sus respectivas tablas (Tablas 3.9 a 3.14).

Tabla 3.9: Probabilidad *Logs* >40 x minuto

Ataque / **>10xMin**	**0**	**1**	**2**	**3**
0	0.8	0.3	0.33	0.28
1	0.2	0.7	0.66	0.71

Tabla 3.10: Probabilidad de *Logs* >2 x segundo

Ataque / **>2xSeg**	**0**	**1**	**2**	**3**
0	0.8	0.66	0.33	0.42
1	0.2	0.4	0.66	0.57

Tabla 3.11: Probabilidad de Match MAC o IP

Ataque / **Match**	**0**	**1**	**2**	**3**
0	0.9	0.5	0.88	0.85
1	0.1	0.5	0.11	0.14

Tabla 3.12: Probabilidad de Tipo de MAC

Tipo MAC \ Ataque	0	1	2	3
0	0.63	0.81	0.5	0.375
1	0.18	0.09	0.3	0.25
2	0.18	0.09	0.2	0.375

Tabla 3.13: Probabilidad de Banderas TCP

Banderas \ Ataque	0	1	2	3
0	0.38	0.38	0.41	0.5
1	0.15	0.23	0.83	0.1
2	0.15	0.15	0.16	0.1
3	0.15	0.15	0.16	0.1
4	0.15	0.07	0.16	0.2

Tabla 3.14: Probabilidad de Ataques

Ataque			
0	1	2	3
0.28	0.28	0.25	0.17

La implementación del algoritmo se realizó en el lenguaje de programación *Python*, la jerarquía del algoritmo se muestra en el apéndice B.

El proceso de ejecución para poner en marcha el funcionamiento del NIDS y la implementación del Algoritmo como tal es la siguiente:

1. **Primer paso de ejecución**

   a) Ponemos la interfaz de red en modo de captura.

   b) Lanzamos el paquete *Discover* para detectar posibles servidores DHCP intrusos.

   c) Analizamos el archivo de *Logs* que genera el Switch para detectar si existe un ataque SYN *Flood*.

2. **Segundo paso de ejecución**

   a) Creamos los Filtros para el Algoritmo.

b) Ejecución del Algoritmo.

c) Creación y apertura del Reporte de resultados.

d) Actualización del archivo de *Logs*.

3.2.1. Primer paso de ejecución

Interfaz en Modo Captura

El primer paso que hace el programa es poner la interfaz de red en modo captura utilizando la herramienta *TCPDUMP*[5]:

```
tcpdump -i eth0 -n -c 10 'tcp or (port 67 or port 68)' -w ./Bayes/lectura.pcap &
```

La interfaz predeterminada es la `eth0`, evitamos la resolución de nombres de protocolos con la opción `-n` y capturamos solamente 20 paquetes, ya que se determinó que eran los suficientes para detectar los ataques. También filtramos el tráfico por protocolos DHCP y TCP y guardamos la captura (`lectura.pcap`) para su posterior filtrado para el algoritmo y dejamos el programa en segundo plano.

Paquete Discover

Ahora lanzamos una serie de 5 paquetes *Discover* utilizando la herramienta *Scapy*[6]

```
scapy -c ./Scapy/dhcp_request.py >/dev/null
```

Le indicamos a *Scapy* que tome las instrucciones del archivo `dhcp_request.py`, al lanzar éstos paquetes, obligamos a cualquier servidor presente en la red a que nos mande un paquete *Offer*, y de ésta manera obtener los parámetros de éstos paquetes para verificar si provienen de nuestro servidor legítimo o de un servidor intruso.

Archivo de *Logs*

Ahora analizamos el archivo de *Logs* que genera el Switch.

```
python Logs.py
```

En este paso, determinamos si existe lo que antes mencionamos sobre los *Logs*, es decir, si hay una cantidad mayor de 10 x minuto y de 2 x segundo, de ser así, enviamos una serie de *pings* a la dirección IP que está siendo atacada para determinar si responde o ya se logró el ataque SYN *Flood* en la víctima. En caso de que no haya presencia de *Logs*, enviamos los *pings* a la dirección de nuestro servidor (Ver tabla 3.2.1).

[5] Ver apéndice A.5 para instalación y más información

[6] Ver apéndice A.6 para instalación y más información.

Tabla 3.15: Fragmento de Código Bayes.py, Análisis de *Logs*.

```
#Log[0] -> 10 x Min
#Log[1] -> 2 x Seg
if Log[0] == "1" or Log[1] == "1":
  print ">> Lanzando Ping a :",IP_Src,"\n"
  os.system('hping3 -c 5 %s'%IP_Src)
else:
  print ">> Lanzando Pings a: ",ip_servidor,"\n"
  os.system('hping3 -c 5 %s'%ip_servidor)
```

3.2.2. Segundo paso de ejecución

Filtrado de captura de paquetes

Después de que tenemos la captura del tráfico en el archivo `captura.pcap`, ahora vamos a filtrar sólo la información específica que necesita el algoritmo. Utilizando la herramienta *tshark*[7], vamos a crear tres filtros, el primero (archivo: `dhcp.txt`) será para filtrar tráfico **DHCP**, y obtenemos los parámetros **IP , MAC y puerto fuente e IP, MAC y puerto de destino:**

```
tshark -n -r ./Bayes/lectura.pcap -T fields -E separator=, -e ip.src -e eth.src
-e udp.srcport -e ip.dst -e eth.dst -e udp.dstport
'(udp.srcport==68 or udp.dstport==68)' > ./Bayes/Filtro/dhcp.txt
```

Ahora filtramos el tipo de mensaje DHCP y lo guardamos en `dhcp_type.txt` con la siguiente instrucción:

```
tshark -n -r ./Bayes/lectura.pcap -o column.format:""Inf","\%i""
'(udp.srcport==68 or udp.dstport==68)' > ./Bayes/Filtro/dhcp_type.txt
```

Para alimentar al algoritmo también necesitamos filtrar la información de los paquetes **TCP**, al igual que con los paquetes DHCP, filtramos **IP y MAc fuente e IP y MAC destino,** además las banderas que antes mencionamos **SYN, ACK, Reset, Push y Fin.** Guardamos el resultado en el archivo: `tcp.txt`, el filtro es el siguiente:

```
tshark -n -r ./Bayes/lectura.pcap -T fields -E separator=,
-e ip.src -e eth.src -e ip.dst -e eth.dst -e tcp.flags.syn
-e tcp.flags.ack -e tcp.flags.reset -e tcp.flags.push
-e tcp.flags.fin  '(tcp)' > ./Bayes/Filtro/tcp.txt
```

[7] Ver apéndice A.4 para instalación y más información

Ejecución del Algoritmo:

Una vez que se tienen los archivos con la información filtrada que se requiere, se ejecuta el algoritmo, y el procedimiento de ejecución es el siguiente:

- Primero creamos un sólo Filtro DHCP con los dos archivos generados con la herramienta *Tshark*, donde se tenga la información de los paquetes DHCP y el tipo de cada uno de ellos. (Tabla 3.16):

Tabla 3.16: Fragmento de Filtro_DHCP.py, Filtro DHCP del Algoritmo.

```
19 def CrearEntradaAlg():
20   #Apertura de archivos
21   try:
22     arch_dhcp = open("./Bayes/Filtro/dhcp.txt")
23   except:
24     print "Error al abrir archivo: dhcp.txt"
25
26   try:
27     entrada_dhcp_alg = open("./Bayes/Filtro/entrada_dhcp_alg.txt","
         w")
28   except:
29     print "Error al abrir archivo: entrada_dhcp_alg.txt"
30
31   lin_dhcp = arch_dhcp.readlines()
32   if lin_dhcp == []:
33     print ">> Vacio : dhcp.txt"
34   else:
35   #Obtenemos el tipo
36   tipo = ObtenerTipoDhcp("./Bayes/Filtro/dhcp_type.txt")
37   i=0
38   for elemento in lin_dhcp:
39     elem_sn_salto=elemento.partition("\n")
40     lin_arch_fin = elem_sn_salto[0]+","+tipo[i]+"\n"
41     i=i+1
42     #Agregamos el tipo al archivo
43     entrada_dhcp_alg.write(lin_arch_fin)
44
45   arch_dhcp.close()
46   entrada_dhcp_alg.close()
47   print ">> Creado: entrada_dhcp_alg.txt"
```

- Generamos la Tabla de Conteos (Tabla 3.17) y la Tabla de Probabilidades (Tabla 3.18):

Tabla 3.17: Fragmento de Código Conteo.py, Tabla de Conteos.

```
.
.
.
def Conteo():
  #Obtenemos los datos de la Tabla de aprendizaje
  try:
    #Leemos archivo tabla_aprend.txt
    tabla = open ("./Bayes/Conteos/tabla_aprend.txt")
    #Creamos la tabla_conteos.txt
    t = open("./Bayes/Conteos/tabla_conteos.txt","w")
  except:
    print "Error al abrir archivo: tabla_aprend.txt"

  lineas_tabla = tabla.readlines()
  Log_Min  = C_Log_Min.C_Log_Min(lineas_tabla)
  log_Seg  = C_Log_Seg.C_Log_Seg(lineas_tabla)
  Match    = C_Match.C_Match(lineas_tabla)
  Tipo_MAC = C_Tipo_MAC.C_Tipo_MAC(lineas_tabla)
  SYN      = C_SYN.C_SYN(lineas_tabla)
  Ataque   = C_Ataque.C_Ataque(lineas_tabla)

  linea = str(Log_Min[0]) + "," + str(Log_Min[1]) + "," + str(
     Log_Min[2]) + "," + str(Log_Min[3]) + ","
  linea = linea + str(Log_Min[4]) + "," + str(Log_Min[5]) + "," +
     str(Log_Min[6]) + "," + str(Log_Min[7]) + "\n"
  t.write(linea)

  .
  .
  .
  print ">> Creado: tabla_Conteos.txt"
  tabla.close()
  t.close()
```

Tabla 3.18: Fragmento de Código Probabilidad.py, Tabla de Probabilidades.

```
.
.
.
def Probabilidad ():
  try:
    #Abrimos la tabla de Conteos
    tabla_c = open ("./Bayes/Conteos/tabla_conteos.txt")
  except:
    print "Error al abrir archivo: tabla_conteos.txt"
  try:
    #Creamos la tabla de probabilidades
    t = open ("./Bayes/Probabilidades/tabla_probabilidades.txt","w"
        )
  except:
    print "Error al abrir archivo: tabla_probabilidades.txt"

  lineas = tabla_c.readlines()
  Log_Min  = P_Log_Min.P_Log_Min(lineas)
  Log_Seg  = P_Log_Seg.P_Log_Seg(lineas)
  Match          = P_Match.P_Match(lineas)
  Tipo_MAC = P_Tipo_MAC.P_Tipo_MAC(lineas)
  SYN    = P_SYN.P_SYN(lineas)
  Ataque         = P_Ataques.P_Ataques(lineas)

  linea = str(Log_Min[0]) + "," + str(Log_Min[1]) + "," + str(
     Log_Min[2]) + "," + str(Log_Min[3]) + ","
  linea = linea + str(Log_Min[4]) + "," + str(Log_Min[5]) + "," +
     str(Log_Min[6]) + "," + str(Log_Min[7]) + "\n"
       t.write(linea)
  .
  .
  .
  print ">> Creado: tabla_Pabilidades.txt"
  tabla_c.close()
  t.close()
```

- Ahora creamos las Instancias para el Algoritmo (Ver Tabla 3.19) , éstas instancias están creadas a partir de la información de los paquetes y del archivo de *Logs*, cada instancia está compuesta por los valores de cada uno de los atributos vistos anteriormente. Ejemplo: Si se tiene la **Instancia: 10101** quiere decir que los valores de los Atributos son: **>10 *Logs* x min. = 1, >2 *Logs* x seg. = 0, Match = 1, Tipo de MAC =**

0 y Banderas = 1.

Tabla 3.19: Fragmento de Código Bayes.py, Creación de instancias.

```
38 #Creamos las instancias para el Algoritmo
39 Ins = Instancia.Crear_Instancia.Crear_Instancia()
40 #Ins[0] = "0/1"
41 #Ins[1] = "0/1"
42 #Ins[2] = [["0/1","ip_src","mac_src","tipo"]]
43 #Ins[3] = [["0/1","ip_src","mac_src"]]
44 #Ins[4] = [["1/2/3/4","ip_src","mac_src","syn","ack","push","rst",
45             "fin"]]
46 #Ins[5] = IP_Log
47 #Ins[6] = MAC_Log
```

Los valores Ins[5] e Ins[6], contienen la IP y la MAC de la PC que está bajo un ataque *SYN Flood.*

- Una vez que tenemos el valor de cada una de las Instancias, es momento de que el Algoritmo realice el cálculo de las probabilidades, en busca de un posible ataque (ver Tabla 3.20):

Tabla 3.20: Fragmento de Código Bayes.py, Prueba de instancias.

```
47 #Probamos las instancias creadas
48 Res = Instancia.Probar.Probar_Instancia.Probar_Instancia(Ins)
49 #Res[] = [["Ins_DHCP"],["P_V_I_DHCP"],["Ins_TCP"],["P_V_I_TCP"]]
50 #P_V_I: Probabilidad del Valor de la Instancia
51 #V_Ins: Valor de la Instancia
```

Res[] contiene una lista del resultado de cada una de las Instancias probadas, el cálculo de éstas, se hizo para los paquetes de tipo DHCP y TCP por separado, compartiendo sólo el valor de ciertos atributos, que nos permiten determinar si se está bajo un ataque simultáneo.

Creación y Apertura del Reporte

El reporte se realiza en LaTeXy se crea un archivo PDF con los resultados. Para generar el reporte, se analizan los resultados tanto de DHCP como de TCP, y determinamos la probabilidad *mayor*, la cual nos dice que de esa instancia específica, se determinó si hay o no un ataque, si existe un posible ataque ya sea de tipo DHCP o TCP, o si hay un ataque pero no puede determinarse de qué tipo es. Todo este procedimiento es automático y lo genera el

código `reporte.py`.

El proceso para el análisis de los resultados de DHCP se realizó de tal manera que se tuviera un balance entre los 4 posibles resultados que arroja el algoritmo y los 6 posibles resultados finales que contendrá el reporte (Ver Tabla 3.21).

Tabla 3.21: Fragmento de Código resultados.py, Análisis de Resultados DHCP.

```
136 if mayor == "SI":
137   msj = "Ataque DHCP Detectado"
138   Escribe_Tex_DHCP.Escribe_Tex_DHCP(i, consola, Instancia_dhcp, ip,
          mac, tipo, pSI, pNO, pPos, pOtro, msj)
139
140   SI_DHCP = SI_DHCP + 1;
141
142 elif mayor == "Posible":
143   msj = "Posible Ataque DHCP Detectado"
144   Escribe_Tex_DHCP.Escribe_Tex_DHCP(i, consola, Instancia_dhcp, ip,
          mac, tipo, pSI, pNO, pPos, pOtro,msj)
145
146   SI_DHCP  = Pos_DHCP + 1;
147
148 elif mayor == "Otro":
149   msj = "Ataque de tipo DHCP Desconocido"
150   Escribe_Tex_DHCP.Escribe_Tex_DHCP(i, consola, Instancia_dhcp, ip,
          mac, tipo, pSI, pNO, pPos, pOtro,msj)
151
152   Pos_DHCP = Pos_DHCP + 1;
153 else:
154   msj = "No se detectaron Ataques"
155   Escribe_Tex_DHCP.Escribe_Tex_DHCP(i, consola, Instancia_dhcp, ip,
          mac, tipo, pSI, pNO, pPos, pOtro,msj)
```

El procedimiento para los resultados de TCP es análogo al de DHCP, cada parámetro obtenido en los resultados, Probabilidades, IP y MAC de fuente, tipo de paquete para DHCP y banderas para TCP, se escriben en un archivo *.tex* para generar el reporte de resultados.

Para dar una visión general del resultado de todos los paquetes, se realizó un análisis probabilístico posterior, el cual nos facilita la comprensión de la conclusión final, este análisis posterior se explica a continuación.

El reporte entregara 4 probabilidades generales (4 resultados), las cuales son:

- **SÍ hay un ataque DHCP**
- **SÍ hay un ataque TCP**
- **Hay un posible ataque DHCP, TCP o uno Desconocido**
- **NO hay ataques Ni DHCP, Ni TCP**

En base a estas probabilidades, se entregan 6 posibles resultados que determinan el estado de la red, lo resultados son:

- **Existe un ataque de tipo DHCP en la red.**
- **Existe un ataque de tipo TCP en la red.**
- **Existen ataques de tipo DHCP y TCP en la red.**
- **Existe un posible ataque en la red.**
- **No existen ataques en la red.**
- **Los resultados no son claros y se debe analizar con detalle.**

En caso de que los resultados no otorguen un panorama claro de lo que está pasando, el reporte contendrá el mensaje *"LOS RESULTADOS NO SON CLAROS Y SE DEBE ANALIZAR CON DETALLE"*, debido a este caso, se incluyen también en el reporte, todos los resultados obtenidos para cada análisis, tanto DHCP como TCP, y se deja a criterio del administrador de la red, el determinar la conclusión final en base a lo que se observe en los resultados completos.

Actualización del archivo de *Logs*

Una vez que el algoritmo termina su ejecución, el archivo de *Logs* tiene que actualizarse, ya que ante cada ejecución del algoritmo, se requieren datos actualizados de lo que está sucediendo en la red, para ello agregamos una cadena vacía al archivo de *Logs*. Este procedimiento es el último paso del segundo proceso de ejecución del algoritmo, y se logra de la siguiente manera:

```
NIDS\#echo > /var/log/cisco.log ‘‘’’
```

Todo este procedimiento explicado hasta ahora es automático, y se logra ejecutando el archivo `run` (tabla 3.22), el cual otorga permisos de ejecución al archivo `run_1` (tabla 3.23) y al archivo `run_2` (tabla 3.24), los cuales ejecutan el primer y segundo paso de ejecución respectivamente. El archivo `run` se ejecuta de la siguiente manera:

Primero accedemos como `root` y damos permisos de ejecución al archivo `run` y después lo ejecutamos.

```
NIDS~#chmod +x ./run
NIDS~#./run
```

NOTA: Antes de ejecutar el archivo `./run`, se debe asegurar en los siguientes archivos `.py`, se tenga la ruta correcta donde se encuentra la carpeta que contiene el algoritmo, es decir la carpeta `NIDS`, ya que el archivo de *Logs* se copiara en esa ruta. Debe de colocar la ruta correcta en los siguientes archivos:

- **./NIDS/Logs.py:** Colocar la ruta correcta en la linea 6:
  `os.system('cp /var/log/cisco.log  /Desktop/NIDS/cisco.log')`
- **./NIDS/Bayes/Instancia/Logs.py:** Colocar la ruta correcta en la linea 6:
  `os.system('cp /var/log/cisco.log  /Desktop/NIDS/cisco.log')`

Además en el archivo `./NIDS/Bayes/Instancia/MatchMAC_IP.py`, debe colocar la MAC e IP de su servidor, en las lineas 8 y 9:

```
8 ip_servidor="192.168.1.253"

9 mac_servidor="00:23:33:23:b3:20"
```

Tabla 3.22: Archivo `run`

```
1 #Archivo de Inicio
2 chmod 777 run_1
3 chmod 777 run_2
4 ./run_1
5 ./run_2
```

Tabla 3.23: Archivo `run_1`, Primer paso de ejecución

```
1  #Primer paso del procedimiento
2  echo -e "\n>> Escuchando Interfaz de red ..."
3  tcpdump -i eth0 -n -c 20 'tcp or (port 67 or port 68)' -w ./Bayes/
       lectura.pcap &
4
5  echo -e "\n>> Lanzando Discover ...\n"
6  scapy -c ./Scapy/dhcp_request.py >/dev/null
7
8  echo ">> Analizando Logs ..."
9  python Logs.py
10
11 wait
12 wait
```

Tabla 3.24: Archivo `run_2`, Segundo paso de ejecución

```
#Segundo paso del procedimiento de ejecucion

echo -e "\n>> Creando filtro dhcp ...\n"
tshark -n -r ./Bayes/lectura.pcap -T fields -E separator=, -e ip.src -e eth.src -e udp.srcport -e ip.dst -e eth.dst -e udp.dstport '(udp.srcport==68 or udp.dstport==68)' > ./Bayes/Filtro/dhcp.txt

tshark -n -r ./Bayes/lectura.pcap -o column.format:""Inf","%i"" '(udp.srcport==68 or udp.dstport==68)' > ./Bayes/Filtro/dhcp_type.txt

echo -e "\n>> Creando filtro tcp ...\n"

tshark -n -r ./Bayes/lectura.pcap -T fields -E separator=, -e ip.src -e eth.src -e ip.dst -e eth.dst -e tcp.flags.syn -e tcp.flags.ack -e tcp.flags.reset -e tcp.flags.push -e tcp.flags.fin '(tcp)' > ./Bayes/Filtro/tcp.txt


echo -e "\n>> Ejecutando Algoritmo ..."
python ./Bayes/Bayes.py

echo ">> Creando reporte ..."
python ./Reporte/reporte.py
pdflatex Reporte/reporte.tex >errores

echo ">> Abriendo reporte ..."
evince reporte.pdf &

echo -e "\n>> Actualizando archivo de Logs ..."
echo > /var/log/cisco.log ""
```

Capítulo 4

Resultados

4.1. Ejemplo de Prueba

Antes de presentar los resultados generales obtenidos, se mostrará un ejemplo de cómo calcular una instancia enfocada a nuestro problema, para mostrar un posible resultado que se pueda presentar.

Calcularemos cuál es el resultado de probar la **Instancia: 11001**, que se obtuvo al analizar un paquete TCP en las pruebas realizadas. El procedimiento que seguiremos es el siguiente, primero debemos obtener las probabilidades de cada una de las combinaciones del valor de los atributos de la instancia con respecto al valor del ataque, es decir, debemos obtener (A=Ataque, LM = LogxMin, LS = LogxSeg, M = Match, TM = Tipo Mac, B = Bandera):

- P(A=No) = P(LM=1|A=0)P(LS=1|A=0)P(M=0|A=0)P(TM=0|A=0)P(B=1|A=0)*P(A=0)
- P(A=Si) = P(LM=1|A=1)P(LS=1|A=1)P(M=0|A=1)P(TM=0|A=1)P(B=1|A=1)*P(A=1)
- P(A=Pos) = P(LM=1|A=2)P(LS=1|A=2)P(M=0|A=2)P(TM=0|A=2)P(B=1|A=2)*P(A=2)
- P(A=Otro)= P(LM=1|A=3)P(LS=1|A=3)P(M=0|A=3)P(TM=0|A=3)P(B=1|A=3)*P(A=3)

Ahora tomando los valores de las tablas de probabilidades antes calculadas, tenemos:

- $P(A = No) = (0{,}2)(0{,}2)(0{,}9)(0{,}63)(0{,}15) * (0{,}28) = 0{,}000952$
- $P(A = Si) = (0{,}7)(0{,}4)(0{,}5)(0{,}81)(0{,}23) * (0{,}28) = 0{,}007302$
- $P(A = Pos) = (0{,}66)(0{,}66)(0{,}88)(0{,}5)(0{,}08) * (0{,}25) = 0{,}003833$
- $P(A = Otro) = (0{,}71)(0{,}57)(0{,}85)(0{,}375)(0{,}1) * (0{,}17) = 0{,}002192$

Y ahora sólo normalizamos cada valor para obtener el resultado final, utilizamos la fórmula $PA = (PA/PA + PB + PC + PD) * 100$:

- P(A=No) = 6.66 %
- P(A=Si) = 51.13 %
- P(A=Pos) = 26.84 %
- P(A=Otro)= 15.35 %

Notamos que la probabilidad más alta es **Sí = 51.13 %**, y de acuerdo a la instancia **11001** el resultado es correcto, pues ambos *Logs* están activos y la Bandera TCP = 1, dándonos la certeza de que la red está bajo un ataque TCP.

4.2. Escenario Completo

Para determinar la eficiencia del Algoritmo y del NIDS en general se llevaron a cabo 10 pruebas para cada posible situación que se pudiera presentar durante el análisis del NIDS. Es decir, se ejecutó el programa en 4 distintos escenarios, cada uno por separado para lograr resultados veraces. Los 4 escenarios son:

1. Ningún ataque presente.
2. Ambos ataques presentes (DHCP *Spoofing* y TCP *SYN Flood*).
3. Sólo el ataque DHCP *Spoofing* presente.
4. Sólo el ataque TCP SYN *Flood* presente.

Para cada uno de los escenarios se pudieron obtener los 6 distintos resultados antes mencionados:

- Existe un ataque de tipo DHCP en la red.
- Existe un ataque de tipo TCP en la red.
- Existen ataques de tipo DHCP y TCP en la red.
- Existe un posible ataque en la red.
- No existen ataques en la red.
- Los resultados no son claros y se debe analizar con detalle.

Las pruebas se realizaron con una lectura de 20 paquetes cada una, debido a que la eficiencia del algoritmo es buena, y se determinó que con tan sólo 20 paquetes se podía tener la suficiente información recabada para otorgar resultados fiables.

En total se realizaron 40 pruebas, 10 para cada escenario, los resultados se muestran a en las tablas 4.1, 4.2, 4.3 y 4.4.

4.3. Análisis y Discusión de Resultados

Se puede observar que en el Escenario 3, donde sólo está presente el ataque DHCP *Spoofing*, los resultados finales no otorgan lo que realmente está ocurriendo en la red, esto es debido a que cuando se analizan los paquetes DHCP también se analizan los paquetes TCP, para cubrir el caso de que los dos ataques estén presentes, entonces cuando sólo está presente el ataque DHCP *Spoofing*, la probabilidad de que NO haya ataque TCP es del 100 %, lo que hace que en los resultados finales impacte de tal forma, que se obtenga que NO hay ataques en la red.

El resultado de este escenario en especial no es tan alarmante gracias a que se muestran todos los paquetes analizados en el reporte, y es ahí donde se ve claramente que existe una cantidad considerable de paquetes ***Offer*** con dirección IP y MAC distintas a la de nuestro servidor legítimo, lo que es un argumento suficiente para decir que **Hay un ataque DHCP**, y darle menos importancia al resultado entregado en el reporte.

En la tabla 4.4 que muestra el escenario donde sólo el ataque TCP está presente, notamos que existe un 25 % de ataques de DHCP, esto es debido a que cuando se analiza un cierta instancia de DHCP (11000), el algoritmo entrega un valor erróneo, y es justamente esa instancia la que se presenta cuando sólo el ataque TCP está presente y es por ello que existe ese porcentaje presente en los resultados, dejando así el porcentaje de un ataque TCP en un 75 %.

Se puede observar en las tablas 4.1 y 4.2, que la primera prueba realizada varía con respecto a las otras 9, esto es debido a que en las primeras pruebas, el archivo de *Logs*, aun contiene información de ataques anteriores (o ninguno), lo que influye en el valor de los atributos de *Logs* y por lo tanto en las probabilidades. Generalmente puede o no presentarse esta situación, todo dependerá de qué escenario este antes o después de cada prueba realizada, pero normalmente, cuando se realizan más de una prueba continua bajo el mismo escenario, los resultados son confiables en un 95 % aproximadamente.

Tabla 4.1: Resultado de Prueba, Escenario 1

Escenario 1: **Ningún Ataque Presente**					
# Prueba	Probabilidades				Resultado Final
	Ataque DHCP	Ataque TCP	Posible Ataque	No Ataques	
1	30 %	35 %	35 %	0 %	Resultados No Claros
2	0 %	0 %	0 %	100 %	No Existen Ataques
3	0 %	0 %	0 %	100 %	No Existen Ataques
4	0 %	0 %	0 %	100 %	No Existen Ataques
5	0 %	0 %	0 %	100 %	No Existen Ataques
6	0 %	0 %	0 %	100 %	No Existen Ataques
7	0 %	0 %	0 %	100 %	No Existen Ataques
8	0 %	0 %	0 %	100 %	No Existen Ataques
9	0 %	0 %	0 %	100 %	No Existen Ataques
10	0 %	0 %	0 %	100 %	No Existen Ataques

Tabla 4.2: Resultado de Prueba, Escenario 2

Escenario 2: **Ambos Ataques Presentes**					
# Prueba	Probabilidades				Resultado Final
	Ataque DHCP	Ataque TCP	Posible Ataque	No Ataques	
1	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
2	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
3	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
4	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
5	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
6	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
7	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
8	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
9	50 %	50 %	0 %	0 %	Ambos Ataques Detectados
10	50 %	50 %	0 %	0 %	Ambos Ataques Detectados

Tabla 4.3: Resultado de Prueba, Escenario 3

Escenario 3: **Sólo Ataque DHCP Presente**					
# Prueba	Probabilidades				Resultado Final
	Ataque DHCP	Ataque TCP	Posible Ataque	No Ataques	
1	50 %	50 %	0 %	0 %	Ambos Detectados
2	25 %	0 %	0 %	75 %	No Existen Ataques
3	25 %	0 %	0 %	75 %	No Existen Ataques
4	25 %	0 %	0 %	75 %	No Existen Ataques
5	25 %	0 %	0 %	75 %	No Existen Ataques
6	25 %	0 %	0 %	75 %	No Existen Ataques
7	25 %	0 %	0 %	75 %	No Existen Ataques
8	25 %	0 %	0 %	75 %	No Existen Ataques
9	25 %	0 %	0 %	75 %	No Existen Ataques
10	25 %	0 %	0 %	75 %	No Existen Ataques

Tabla 4.4: Resultado de Prueba, Escenario 4

Escenario 4: **Sólo Ataque TCP Presente**					
# Prueba	Probabilidades				Resultado Final
	Ataque DHCP	Ataque TCP	Posible Ataque	No Ataques	
1	25 %	75 %	0 %	0 %	Ataque TCP Detectado
2	25 %	75 %	0 %	0 %	Ataque TCP Detectado
3	25 %	75 %	0 %	0 %	Ataque TCP Detectado
4	25 %	75 %	0 %	0 %	Ataque TCP Detectado
5	25 %	75 %	0 %	0 %	Ataque TCP Detectado
6	25 %	75 %	0 %	0 %	Ataque TCP Detectado
7	25 %	75 %	0 %	0 %	Ataque TCP Detectado
8	25 %	75 %	0 %	0 %	Ataque TCP Detectado
9	25 %	75 %	0 %	0 %	Ataque TCP Detectado
10	25 %	75 %	0 %	0 %	Ataque TCP Detectado

Capítulo 5

Conclusiones

El algoritmo de Naive Bayes, depende mucho de los atributos y de la Tabla de Aprendizaje. Con respecto a la tabla, entre más grande sea el número de instancias mejor es la precisión de los resultados en términos probabilísticos. Los atributos analizados para la resolución de este problema en particular fueron los adecuados, en cambio la tabla de aprendizaje se tuvo que limitar a 28 instancias, debido a que son todos los casos posibles que pudieran presentarse en cualquiera de los escenarios presentados, y esto impactó en ciertos resultados de algunas instancias como la que se presenta en el escenario 4 (sólo ataque TCP presente). Para solucionar este detalle se modificó la tabla de aprendizaje y se cambiaron algunos valores de atributos, aumentando en un 10 % la certeza del resultado aproximadamente.

Cuando se analiza un paquete DHCP los principales parámetros en los cuales se pone atención son el *tipo de paquete,* IP y MAC. Cuando se quiere detectar un ataque de tipo DHCP *Spoofing*, el parámetro más importante es el *tipo de paquete* ya que, si es de tipo *Offer*, nos dice que existe un servidor presente en la red y podemos analizar si es o no nuestro servidor. En el caso del escenario 3, (sólo el ataque DHCP presente) los resultados nos dicen que hay un 25 % de probabilidad de que haya un ataque DHCP, y si analizamos los resultados completos[1], nos damos cuenta que en efecto, estamos detectando paquetes Offer de una dirección IP y MAC que no son la de nuestro servidor. Es decir cuando se detecta un paquete *Offer* con parámetros distintos a los de un servidor legítimo, es argumento suficiente para asegurar que se está bajo un ataque DHCP *Spoofing*, lo que nos permite descartar el resultado general que entrega el NIDS.

Existen algunos ataques que no necesariamente deben de tener un patrón determinado para su detección, es decir, requieren de un análisis más a detalle de parte del administrador de la red para su detección. Basándonos en esto, podemos decir que cuando se utiliza una base de firmas para detectar los ataques mencionados, se está haciendo un gasto computacional

[1] Ir al apéndice C para ver ejemplo de resultado obtenido.

innecesario. En nuestro caso nuestra base de firmas son 28 instancias las que nosotros construimos para enseñarle algoritmo como actuar bajo distintos escenarios, lo que nos permite tener un gasto computacional bastante bajo y por consiguiente un rendimiento computacional muy bueno.

Un NIDS como ya lo mencionamos anteriormente, requiere del apoyo de equipos de comunicación (Switch, Router, etc.), para la solución de este proyecto fue bastante necesario el apoyo del Switch, ya que si bien podemos prescindir de el para la detección del ataque DHCP *Spoofing*, no podemos hacerlo para la detección del ataque TCP SYN *Flood*, ya que requerimos el análisis de *Logs*, por lo tanto para cualquier solución similar a un NIDS es necesario contar con estos equipos de comunicación para facilitar la detección de ataques.

5.1. Conclusión General

El desarrollo e implementación del Algoritmo Naive Bayes en el Sistema Embebido es una buena solución, ya que es capaz de detectar los ataques de DHCP *Spoofing* y TCP SYN *Flood*, con la ventaja de que no requiere una gran base de firmas para lograrlo.

Hablando de términos computacionales, a pesar de que se instaló un sistema operativo algo robusto en el sistema embebido y se utilizó además un lenguaje de programación interpretado como lo es Python, el costo computacional es bastante bajo, y el tiempo de ejecución es excelente, al rededor de 8 a 10 segundos como máximo y 5 segundos en promedio.

5.2. Trabajo a futuro

En la actualidad existen varias herramientas comerciales y NIDS's gratuitos como SNORT por ejemplo, pero todas ellas tienen algo en común, y es que la documentación es bastante deficiente, ninguna de ellas menciona detalladamente los algoritmos utilizados, herramientas o metodología con la que funcionan, y este trabajo presenta una gran ventaja ante ello, ya que se presenta la metodología que se utiliza para la detección de los ataques resueltos, además de proporcionar el código fuente del algoritmo que se utiliza, dejando así abierta la posibilidad de que se puedan hacer correcciones o mejoras al código o al sistema en general y se tenga un mayor alcance y un marco de detección de ataques mucho más amplio.

Apéndice A

Herramientas

A.1. *Syslog*

Para recibir los *Logs* del Switch se debe configurar el servicio en el Switch y además, es necesario instalar y configurar el Servidor y Cliente *Syslog* en el NIDS también, las configuraciones se hacen de la siguiente forma:

- **Instalación**

  - Primero agregamos una llave y los repositorios necesarios:

    ```
    NIDS~#apt-key adv --recv-keys --keyserver keys.gnupg.net AEF0CF8E
    gpg --export --armor AEF0CF8E | sudo apt-key add - nano /etc/apt/sources.list
    ```

  - Ahora actualizamos e instalamos la aplicación:

    ```
    NIDS~#apt-get update && apt-get upgrade
    NIDS~#apt-get install rsyslog
    ```

- **Servidor Central de *Logs* (NIDS)**

  - Una vez teniendo la aplicación instalada configuramos el Servidor Central. Primero editamos el siguiente archivo:

    ```
    NIDS~#nano /etc/default/rsyslog
    # Options for rsyslogd
    # -m 0 disables 'MARK' messages (deprecated, only used in compat mode < 3)
    # -r enables logging from remote machines
    #(deprecated, only used in compat mode < 3)
    # -x disables DNS lookups on messages received with -r
    # -c compatibility mode
    # See rsyslogd(8) for more details
    RSYSLOGD_OPTIONS="-c4"
    ```

    Cambiamos "-c4" por "-r514":

    ```
    RSYSLOGD_OPTIONS="-r514"
    ```

- Ahora editamos el archivo */etc/rsyslog.conf* y descomentamos las siguientes lineas:

```
$ModLoad imudp
$UDPServerRun 514
```

En el mismo archivo agregamos los siguiente para configurar el archivo y ruta donde se guardaran los *Logs*

```
# This one is the template to generate the log filename dynamically,
#depending on the client's IP address.
$template FILENAME,"/var/log/%fromhost-ip%/syslog.log"

# Log all messages to the dynamically formed file.
#Now each clients log (192.168.1.2, 192.168.1.3,etc...), will be u$
*.* ?FILENAME
```

- **Cliente *Syslog***

- El procedimiento para configurar el Cliente es similar al del Servidor, sólo agregamos las siguientes lineas al archivo */etc/rsyslog.conf:*

```
# Agregamos ip y puerto del servidor
*.*    @192.168.1.1:514
```

- Ahora sólo reiniciamos el servicio:

```
/etc/init.d/rsyslog restart
```

- **Cliente *Syslog*, Switch CISCO**

- Primero configuramos los parámetros de *Logging*

```
Switch(conf)#loggin on
Switch(conf)#loggin 192.168.1.1
Switch(conf)#loggin trap debug
```

- Y ahora le decimos al Switch, de qué eventos genere *Logs*:

```
Switch#debug mac-notification
Switch#debug ethernet-interface
Switch#debug port-security
Switch#debug arp
Switch#debug dhcp detail
Switch#debug interface fa0/1-24
Switch#debug packet
```

A.2. *Hping3*

Hping3 (la nueva versión de hping), son scripts que usan el lenguaje Tcl e implementan un motor de fácil entendimiento de paquetes TCP/IP, de forma que el programador puede escribir secuencias de comandos relacionados con la manipulación de paquetes TCP/IP de

bajo nivel y el análisis en un tiempo muy corto[11].

Antes de instalar *Hping3*, debe de instalar las siguientes dependencias:

- tcl8.4
- libcap0.8
- libc6

Una vez instaladas las dependencias mencionadas, como usuario `root` agregamos el siguiente repositorio:

```
root#deb http://us.archive.ubuntu.com/ubuntu precise main universe
```

Ahora actualizamos los repositorios:

```
root#apt-get update
```

Finalmente instalamos la herramienta:

```
root#apt-get install hping3
```

A.3. *Ettercap*

Ettercap es una suite completa para ataques MITM. Cuenta con análisis de conexiones en tiempo real, filtrado de contenido sobre la marcha y muchos otros trucos interesantes. Es compatible con la disección activa y pasiva de muchos protocolos e incluye muchas características de la red y el análisis de host[12].

Esta herramienta está preinstalada en la distribución Kali Linux utilizada en este proyecto, pero se explicará su instalación para otras distribuciones de linux.

Primero instalamos como usuario `root` los siguientes paquetes necesarios para el correcto funcionamiento de la herramienta:

```
root#apt-get install build-essential
root#apt-get install linux-headers-‘uname -r‘
```

También necesitamos las siguientes dependencias:

- libpcre3-dev
- libpcap0.8-dev
- libnet1-dev
- openssl
- libssl-dev

- ncurses-bin
- libncurses5-dev
- libnet6-1.3-dev
- libpthread-stubs0-dev
- zlib1g-dev
- libltdl-dev
- pango-graphite
- pkg-config
- libpango1.0-dev
- libatk1.0-dev
- libgtk2.0-dev
- autoconf
- byacc

Ahora descargamos *Ettercap* de su página oficial http://ettercap.github.io/ettercap/downloads.html, descomprimimos el archivo `ettercap-0.8.0.tar.gz` y accedemos a la carpeta descomprimida.

Tecleamos como `root` el siguiente comando: `./autogen.sh`.

Ejecutamos el archivo `.configure` habilitando los plugins:

```
root#./configure --enable-plugins --enable-debug
```

Una vez hecho lo anterior, abrimos el archivo `Makefile` de la carpeta src, `ettercap/src`, y buscamos la línea `LIBS = -lresolv -lz -lpthread -lltdl -ldl -ldl` y la reemplazamos por:

```
LIBS = -lresolv -lz -lpthread -lltdl -ldl -ldl -lpcap -lnet -lssl -lcrypto
-lpcre -lpanel -lmenu -lform -lncurses -pthread -lgtk-x11-2.0 -lgdk-x11-2.0
-lgio-2.0 -lpangoft2-1.0 -lpangocairo-1.0 -lgdk_pixbuf-2.0 -lcairo
-lfreetype -lfontconfig -lpango-1.0 -lgmodule-2.0 -latk-1.0 -lgobject-2.0
-lgthread-2.0 -lrt -lglib-2.0 -lgthread-2.0
```

Ahora sólo ejecutamos `make` y una vez terminado el procedimiento ejecutamos el comando `make install` y eso es todo.

A.4. *Tshark*

Tshark es un analizador de protocolos de red. Permite capturar paquetes de la red en tiempo real, o leer paquetes de un archivo de captura guardado anteriormente, o bien la impresión de una forma decodificada de esos paquetes a la salida estándar o escribiendo los paquetes a un archivo. El formato de archivo de captura nativo de *Tshark* es un formato `pcap`, que es también el formato utilizado por tcpdump y otras herramientas.
Sin ningún conjunto de opciones, *Tshark* trabajará mucho como tcpdump. Se utilizará la biblioteca pcap para capturar el tráfico de la primera interfaz de red disponible y muestra una línea de resumen en la salida estándar para cada paquete recibido[13].

Para instalar esta herramienta, basta con ejecutar el siguiente comando como usuario `root`: `root#apt-get install tshark`

A.5. TCPDUMP

Tcpdump es un potente analizador de paquetes de línea de comandos. Imprime una descripción de los contenidos de los paquetes sobre una interfaz de red que responden a la expresión booleana; la descripción es precedida por un sello de tiempo impreso por defecto, como horas, minutos, segundos y fracciones de segundos desde la medianoche. También se puede ejecutar con la bandera -w, lo que guarda los datos de los paquetes a un archivo para su posterior análisis, y / o con la bandera -r, que hace que se lea desde un archivo guardado de paquetes en lugar de leer los paquetes a partir de una interfaz de red. También se puede ejecutar con la bandera -V, lo que lee una lista de archivos de paquetes guardados[14].

Para instalar esta herramienta, basta con ejecutar el siguiente comando como usuario `root`:

```
root\#apt-get install tcpdump libpcap0.8
```

En caso de que no se encuentre en sus repositorios, podemos instalarla bajando y compilando el código fuente de libpcap y tcpdump, descargamos los archivos comprimidos de sus respectivas páginas oficiales: http://projects/libpcap/yhttp://sourceforge.net/projects/tcpdump/.

Ahora descomprimimos cada uno de los archivos, y en cada una de las carpetas ejecutamos lo siguiente como usuario `root`:

```
./configure
make
sudo make install
```

A.6. *Scapy*

Scapy es una potente aplicación interactiva para la manipulación de paquetes. Es capaz de forjar o decodificar paquetes de un gran número de protocolos, enviarlos en la red, capturarlos y mucho más. Se puede manejar fácilmente la mayoría de las tareas clásicas como el escaneo, *Tracerouting*, el sondeo, ataques o el descubrimiento de la red. También funciona muy bien en muchas de otras tareas específicas que la mayoría de otras herramientas no pueden manejar, como el envío de tramas no válidas o la inyección de sus propios marcos 802.11[15].

Se utilizo ésta herramienta para ayudar al NIDS a realizar la detección del ataque DHCP Spoofing enviando un paquete DHCPDISCOVERY en Broadcast, y de ésta manera recibir los paquetes DHCPOFFER de cualquier servidor ya sea auténtico o no de la red, y de ésta manera darse cuenta si existe o no un servidor falso ya que se conoce a *priori* los datos específicos del servidor DHCP original.

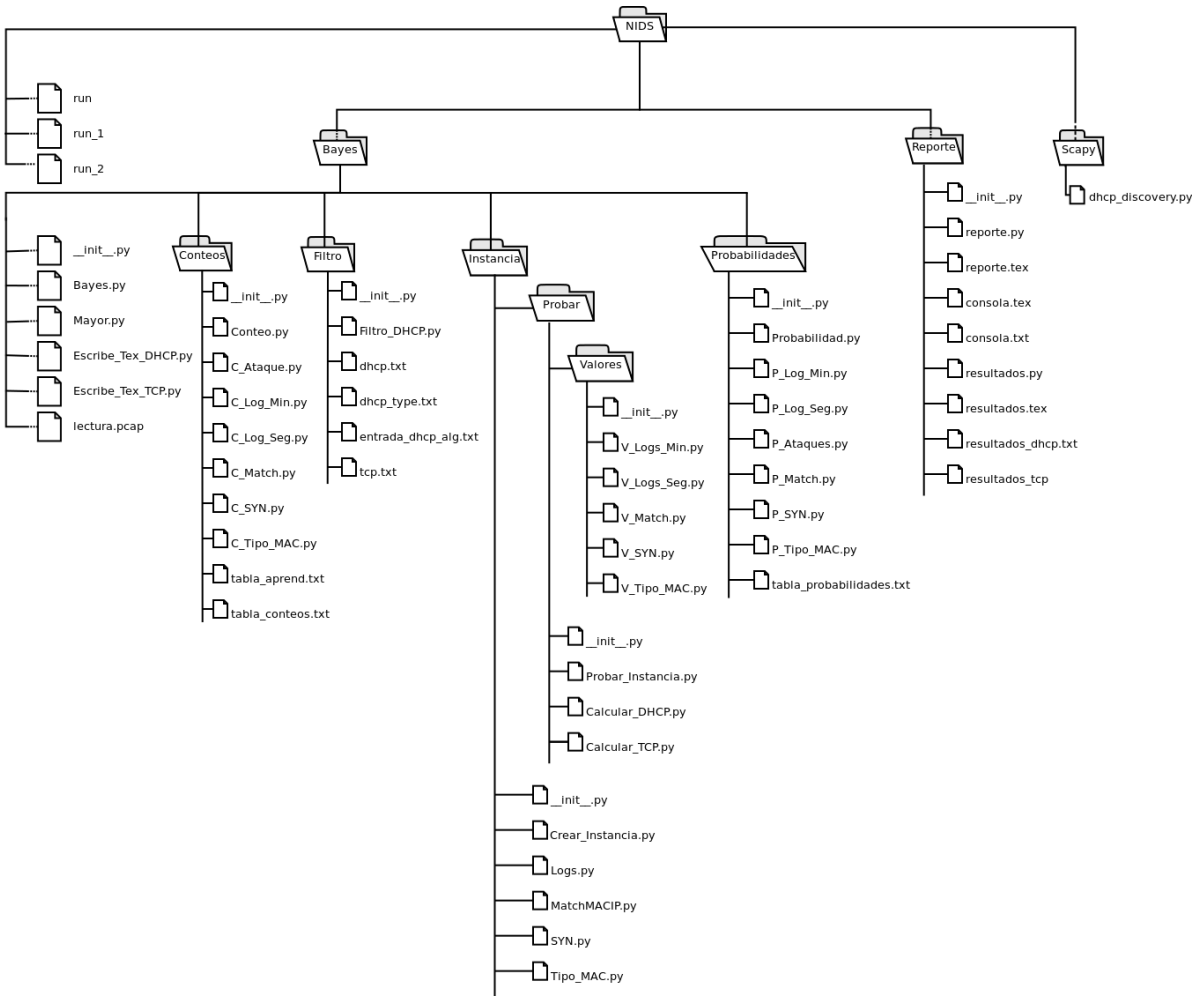
Para instalar esta herramienta basta con seguir los siguientes pasos: Como usuario `root`, ejecutamos lo siguiente:

```
root#wget http://hg.secdev.org/scapy/raw-file/v1.2.0.2/scapy.py
root#python scapy.py
```

Apéndice B

Código Python

El programa realizado en el lenguaje de programación Python se encuentra en el CD que contiene este proyecto en la carpeta ./NIDS. La jerarquía de archivos se muestra a continuación:



Apéndice C

Ejemplos de Resultados Obtenidos

A continuación se presenta un fragmento del reporte obtenido bajo un ataque DHCP (ver figuras C.1 y C.2), los ejemplos completos de los reportes obtenidos bajo cada uno de los escenarios mencionados anteriormente, se encuentran en el CD que contiene el proyecto en la carpeta ./EjemploDeResultados.

Reporter del NIDS

Gibrann Montero Quintero

18 de noviembre de 2014

Resumen

Este reporte contiene los resultados del análisis de tráfico en la red, dichos resultados se presentan de acuerdo al análisis del algoritmo **Naive Bayes**, dando un resultado y diagnóstico aproximados.

1. Resultados de Naive Bayes

Después de realizar el análisis de **20 paquetes** se determinó que:

NO EXISTEN ATAQUES EN LA RED.

Probabilidades generales del análisis de todos los paquetes.

* **Ataque DHCP :** 25.0 %
* **Ataque TCP :** 0.0 %
* **No Ataques :** 75.0 %
* **Posible Ataque:** 0.0 %

Figura C.1: Ejemplo de resultado obtenido bajo un Ataque DHCP, resultado de Naive Bayes.

2. Resultados Completos

- **:: No se detectaron Ataques ::**
  :: Paquete: 1 ::
  - **Instancia:** 00000
  - **IP src:** 0.0.0.0
  - **MAC src:** b8:ac:6f:b5:52:e7
  - **Tipo:** Discover
  - **::: Probabilidades :::**
    - **SI** = 14.1889064346
    - **NO** = 70.6292231414
    - **Posible** = 9.01990000323
    - **Otro** = 6.16197042076
- **:: Ataque DHCP Detectado ::**
  :: Paquete: 2 ::
  - **Instancia:** 00100
  - **IP src:** 192.168.1.12
  - **MAC src:** b8:ac:6f:b5:52:f3
  - **Tipo:** Offer
  - **::: Probabilidades :::**
    - **SI** = 58.6534630191
    - **NO** = 32.4404338674
    - **Posible** = 4.6607571002
    - **Otro** = 4.24534601337
- **:: No se detectaron Ataques ::**
  :: Paquete: 3 ::
  - **Instancia:** 00000
  - **IP src:** 0.0.0.0
  - **::: Probabilidades :::**
    - **SI** = 14.1889064346
    - **NO** = 70.6292231414
    - **Posible** = 9.01990000323
    - **Otro** = 6.16197042076
- **:: Ataque DHCP Detectado ::**
  :: Paquete: 6 ::
  - **Instancia:** 00100
  - **IP src:** 192.168.1.12
  - **MAC src:** b8:ac:6f:b5:52:f3
  - **Tipo:** Offer
  - **::: Probabilidades :::**
    - **SI** = 58.6534630191
    - **NO** = 32.4404338674
    - **Posible** = 4.6607571002
    - **Otro** = 4.24534601337
- **:: No se detectaron Ataques ::**
  :: Paquete: 7 ::
  - **Instancia:** 00000
  - **IP src:** 0.0.0.0
  - **MAC src:** b8:ac:6f:b5:52:e7
  - **Tipo:** Discover
  - **::: Probabilidades :::**
    - **SI** = 14.1889064346
    - **NO** = 70.6292231414
    - **Posible** = 9.01990000323
    - **Otro** = 6.16197042076

Figura C.2: Ejemplo de resultado obtenido bajo un Ataque DHCP, resultados completos.

Bibliografía

[1] Walter Fuertes, Fernando Rodas, Deyci Toscano. "Evaluación de ataques UDP Flood utilizando escenarios virtuales como plataforma experimenta", Revista Facultad de Ingeniería, UPTC, Julio-Diciembre de 2011, Vol. 20, No. 31, pp.37-53.[En linea]. Disponible: http://dialnet.unirioja.es/descarga/articulo/3914182.pdf

[2] Sergio Abuín. *Manipulación de ARP para la intercepción de tráfico.* LINUX- MAGAZINE, Número 29. [En linea]. Disponible: http://www.linux-magazine.es/issue/29/029-034_AtaqueARPLM29.crop.pdf

[3] Jazib Frehim, Omar Santos. *Cisco ASA: All-in-one Firewall, IPS, and VPN Adaptive Security Appliance.* Cisco Press, 2006, p.14.

[4] Sean Convery. *Network Security Architectures.* Cisco Press, 2004, pp.103-109.

[5] AXIOMTEK, "PICO820 Series Intel Atom™All-In-One Pico ITX CPU Board User's Manual", [En linea]. Disponible: http://axiomtek.com/Download/Download/PICO820/PICO820.pdf

[6] Borja Merino, Febrero. "Análisis de Tráfico con Wireshark." Instituto Nacional de Tecnología de la Comunicación, [En linea]. Disponible: http://cert.inteco.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_seguridad_analisis_trafico_wireshark.pdf

[7] P. Ragharan, Amol Lad, Sriram Neelakandan. *Embedded Linux System, Design And Development.* 2006:Averbach Publications, Taylor & Francis Group. pp.1-2.

[8] Mitchell, Tom. *Machine Larning.* McGraw-Hill, 1997, pp. 154-159.

[9] Richard A. Deal. *Cisco Router Firewall Security.* Cisco Press, 2005, pp.92-96.

[10] The MathWorks, Inc. *Machine Larning.* [En linea]. Disponible: http://www.mathworks.com/machine-learning/

[11] Salvatore Sanfilippo. *Hping3.* [En linea]. Disponible: http://linux.die.net/man/8/hping3

[12] Alberto Ornaghi, Marco Valleri. *Ettercap*. [En linea]. Disponible http://ettercap.github.io/ettercap/index.html

[13] Gerald Combs. *Tshark*. [En linea]. Disponible: https://www.wireshark.org/docs/man-pages/tshark.html

[14] Van Jacobson, Craig Leres, Steven McCanne. Universidad de California. *TCPDUMP*. [En linea]. Disponible: http://www.tcpdump.org/index.html

[15] Philippe Biondi. *Scapy*. [En linea]. Disponible http://www.secdev.org/projects/scapy/

[16] Samuel D. Pacheco Leal, Luis Gerardo Díaz Ortiz, Rodolfo García Flores. *El clasificador Naïve Bayes en la extracción de conocimiento de bases de datos*. Posgrado en Ingeniería de Sistemas, FIME-UANL. [En linea]. Disponible www.ingenierias.uanl.mx/27/27_clasificador.pdf